Queries and Computation on the Web*

Serge Abiteboul[1] ** and Victor Vianu[2]

[1] Computer Science Department, Stanford University, Stanford, CA
[2] Univ. of California at San Diego, CSE 0114, La Jolla, CA 92093-0114, USA

Abstract. The paper introduces a model of the Web as an infinite, semistructured set of objects. We reconsider the classical notions of genericity and computability of queries in this new context and relate them to styles of computation prevalent on the Web, based on browsing and searching. We revisit several well-known declarative query languages (first-order logic, Datalog, and Datalog with negation) and consider their computational characteristics in terms the notions introduced in this paper. In particular, we are interested in languages or fragments thereof which can be implemented by browsing, or by browsing and searching combined. Surprisingly, stratified and well-founded semantics for negation turn out to have basic shortcomings in this context, while inflationary semantics emerges as an appealing alternative.

1 Introduction

The World Wide Web [BLCL+94] is a tremendous source of information which can be viewed, in some sense, as a large database. However, the nature of the Web is fundamentally different from traditional databases and raises qualitatively new issues. Its main characteristics are its global nature and the loosely structured information it holds. In this paper, we consider some fundamental aspects of querying the Web.

We use as a model of the Web an abstraction that captures its global nature, and the semistructured information it holds. Perhaps the most fundamental aspect of our model is that we view the Web as infinite. This point of view is not new, and has already been suggested by some researchers [AMM96]. We believe this captures the intuition that exhaustive exploration of the Web is –or will soon become– prohibitively expensive. The infiniteness assumption can be viewed as a convenient metaphor, much like Turing machines with infinite tapes are useful abstractions of computers with finite (but potentially very large) memory. Note that our approach is fundamentally different from previous attempts to model infinite data (e.g. [CH93, KKR90]) which focus on finitely representable

* Work supported in part by the National Science Foundation under grant number IRI-9221268.

** This author's permanent position is INRIA-Rocquencourt, France. His work was supported in part by the Air Force Wright Laboratory Aeronautical Systems Center under ARPA Contract F33615-93-1-1339, and by the Air Force Rome Laboratories under ARPA Contract F30602-95-C-0119

databases. In contrast, we do not assume the Web is finitely represented. Instead, we view it as a possibly nonrecursive infinite structure which can never be entirely explored. Our model leads to a focus on querying and computation where exploration of the Web is controlled. This raises issues akin to safety in classical databases.

The data model we use is similar to several models for unstructured data recently introduced, e.g., [Q+95, CACS94, BDS95]. The Web consists of an infinite set of objects. Objects have a value and/or may reference other objects via labeled links. The set of labels for each object is not fixed, unlike the attributes of a relation. Intuitively, an object can be viewed as a Web page; the value is the content of a page; labels provide links that allow navigating through the Web, in hypertext style.

We begin by exploring the notion of computable query in the context of the Web. Our model is along the lines of the computable queries of Chandra and Harel [CH80]. We introduce a machine model of computation on the Web that we call a Web machine. This works much like a Turing machine, but takes as input an infinite string and may produce an infinite answer. We also introduce two particular machine models that capture directly the main styles of computing used on the Web: browsing and searching. The *browser* machine model allows for navigational exploration of the Web. The *browse/search* machine additionally allows searching in the style of search engines.

Based on the Web machine, we define the notions of computability and eventual computability of queries. The latter notion arises from the fact that infinite answers to queries are allowed. A query is *computable* if its answer is always finite and computable by a halting Web machine. A query is *eventually computable* if there is a Web machine, possibly nonterminating, which eventually outputs each object in the answer to the query. Interesting connections hold with the browser machine and with the browse/search machine. We show that every generic and computable query is in fact computable by a browser machine. This confirms the intuition that browsing is in some sense the only way to control computation on the Web. We also show that the set of generic queries which are eventually computable by a Web machine is precisely the same as the set of generic queries which are eventually computable by a browse/search machine. Thus, everything can be done by a combination of browsing and searching.

To express queries, one needs query languages. We are interested in the ability of *declarative* database query languages to express queries on the Web. To this end, we revisit the classical languages FO (first-order logic), Datalog, and Datalog$^\neg$. The questions of interest for each language are the following: (i) Are the queries in the language computable or eventually computable? (ii) Which fragments of each language can be implemented by browsers and which by a combination of browsing and searching? We provide syntactic restrictions that guarantee computability by browsers or by browse/search machines in FO and Datalog$^{(\neg)}$.

One of the interesting results of the paper is with respect to negation. The "positive" fragment of FO is eventually computable. The addition of recursion yields no problem. However, negation brings trouble, and some simple FO queries are not eventually computable. The Datalog$^\neg$ languages yield some surprises: the standard semantics, stratified and well-founded [GRS88], are ill-suited for expressing eventually computable queries, whereas the more procedural inflationary semantics [AV88, KP88] turns out to be naturally suited to express such queries, and thus has a fundamental advantage over the first two semantics.

Computation on the Web is still in its infancy, and it is premature to propose a definitive model. It is not yet clear what the right abstractions are. We believe that our model of the Web captures some essential aspects; future developments may confirm or invalidate this. Clearly, we have ignored in our investigation many important aspects, such as the communication costs associated with browsing and searching; the notion of locality; the essentially distributed nature of the Web and the fact that concurrent processes may participate in evaluating a query; updates; the fact that users are often satisfied with incomplete, imprecise or partially incorrect answers.

Query languages for the Web have attracted a lot of attention recently, e.g., W3QL [KS95] that focuses on extensibility, WebSQL [AMM96] that provides a formal semantics and introduce a notion of locality, or WebLog [LSS96] that is based on a Datalog-like syntax. Since HTML (the core structure of the Web) can be viewed as an instance of SGML, the work on querying structured document, e.g., [CACS94, GZC89] is also pertinent, along with work on querying semistructured data (see [A97]). The work on query languages for hypertext structures, e.g., [MW95, CM89, MW93] is also relevant.

In the next section, we introduce Web machines, browser machines, and browse/search machines. We then formalize the notion of (eventually) computable query on the Web. The following section considers FO, Datalog and Datalog$^\neg$, and establishes connections to (eventual) computability, browsing, and searching. Finally, we provide some conclusions.

2 Computation on the Web

We model the Web as a set of semistructured objects in the style of [Q$^+$95, BDS95]. More precisely, we view the Web as an infinite database over the fixed relational schema {*Obj(oid), Ref(source,label,destination), Val(oid,value)*}. The meaning of the above relations is as follows:

1. *Obj* provides an infinite set of objects.
2. Relation *Ref* specifies, for some of the objects, a *finite* set of links to other objects, each of which has a label. More precisely, $Ref(o_1,l,o_2)$ indicates that there is an edge labeled l between o_1 and o_2.
3. Relation *Val* specifies a value for some of the objects. Thus, $Val(o,v)$ specifies that object o has value v.

Intuitively, an object corresponds to a Web page. The value is the content of the page, and references model labeled links to other pages.

A *Web instance* is an *infinite* structure[3] over the above schema, satisfying the following constraints:

$Obj = \pi_{source}(Ref) \cup \pi_{oid}(Val)$; *Val* satisfies the fd $source \rightarrow value$;
$\pi_{destination}(Ref) \subseteq Obj$; $\forall o \in Obj$, $\sigma_{source=o}(Ref)$ is finite.

Thus, each object must have a value or some references to other objects. An object can have at most one value and only finitely many references to other objects. Every referenced object must belong to the specified set of objects of the instance. The set of all Web instances is denoted **inst**(Web).

Let I be a Web instance. For each object o in $I(Obj)$, the *description* of o in I consists of the finite set of tuples in I whose first coordinate is o. Thus, the description of an object provides its outgoing links and/or its value. It does *not* provide the set of in-going links (which can be infinite). We may regard *Ref* as a labeled graph whose vertices are objects. We say that object o' is reachable from object o if this holds in the labeled graph given by *Ref*. The distance between two objects is also defined with respect to the *Ref* graph.

A first attempt

We wish to formalize the notion of a query on the Web. We first explore a straightforward extension of the classical notion of query, which we will soon refine. Let a query be a mapping on **inst**(Web) which associates to each Web instance I a subset of $I(Obj)$.

We wish to have a notion of *generic* and *computable* query that is appropriate for the Web. As in the classical definition proposed by Chandra and Harel [CH80], a query is *generic* if it commutes with isomorphisms over **inst**(Web). More precisely, a query q is *generic* if for each I and each one-to-one mapping ρ on the domain (extended to I in the obvious way), $q(\rho(I)) = \rho(q(I))$. Intuitively, this means that the result only depends on the information in I and is independent of any particular encoding chosen for I.

The definition of computability requires a departure from the classical definition, because inputs and outputs are possibly infinite. Let a *Web machine* be a Turing machine with three tapes: (1) a right-infinite input tape, (2) a two-way-infinite work tape, and (3) a right-infinite output tape. Initially, the input tape contains an infinite word (an encoding of the Web instance), and the work and output tapes are empty. The input tape head is positioned at the first cell. The moves are standard, except that the output tape head can only move to the right (so nothing can be erased once it is written on the output tape).

Web instances can be encoded on the input tape in a straightforward manner. Let α be a successor relation on all elements occurring in I (including oid's, labels, and values). For each element e occurring in I, let $enc_\alpha(e)$ be the binary representation of the rank of e in the ordering α. An instance I is encoded as

$$enc_\alpha(\widehat{o_1})\#\#enc_\alpha(\widehat{o_2})\#\#...enc_\alpha(\widehat{o_m})\#\#...$$

[3] All infinite structures mentioned in the paper are countable, unless otherwise specified.

where $o_1, o_2, ..., o_m, ...$ is the list of oid's in $I(Obj)$ in the order specified by α and for each i, $enc_\alpha(\widehat{o_i})$ is a standard encoding with respect to α of the description of o_i. (Recall that the description of o_i is the finite structure.)

Note that in the above encoding, the finite information about each object is clustered together. This has nontrivial consequences. Some of the results below do not hold otherwise. Our encoding presents the advantage that it models accurately the real situation on the Web (information is clustered around pages).

The output $q(I)$ of a query q on input I is a set of objects, that is encoded as $enc_\alpha(o_{i_1})\#\ldots\#enc_\alpha(o_{i_k})\ldots$, where $o_{i_1}, \ldots o_{i_k}\ldots$ are the objects in $q(I)$, in *some* order. No particular order is imposed on the presentation of objects in the answer, so many possible answers are possible. Allowing this flexibility is important for technical reasons, since some of the results below would not hold if we required that objects be output in lexicographical order. (Intuitively, one could not output an object o before being certain that no "smaller" object is in the answer). By slight abuse of notation, we denote any such presentation of the answer by $enc_\alpha(q(I))$.

Let us now make a first attempt at defining the notion of a computable query. A query q is *0-computable* (we will abandon soon this definition) if there exists a Web machine which on input $enc_\alpha(I)$ halts and produces $enc_\alpha(q(I))$ on the output tape, for each I in **inst**(Web) and each α. Note that every 0-computable query produces a finite answer for each input. A query q is *0-eventually computable* if there exists a Web machine whose computation on input $enc_\alpha(I)$ has the following properties:

- the content of the output tape at each point in the computation is a prefix of $enc_\alpha(q(I))$, and
- for each $o \in q(I)$, its encoding $enc_\alpha(o)$ occurs on the output tape at some point in the computation.

Note that if q is 0-eventually computable the Web machine is not required to terminate, even if $q(I)$ happens to be finite.

It turns out that the above definitions need some further refining. Indeed, as things stand, the only queries that are 0-computable are in some sense trivial. More precisely, we call a query q *trivial* if $q(I) = \emptyset$ for every Web instance I. We claim that every 0-computable query is trivial. (Note that there are nontrivial 0-*eventually* computable queries on infinite databases, e.g., the query that outputs the set of oid's.) The argument for 0-computable queries goes as follows. Suppose q is a 0-computable query and $q(I) \neq \emptyset$ for some input I. Let W be a Web machine that computes q. Observe that W only reads a finite prefix ω of $enc_\alpha(I)$. Now consider an instance $\bar{I}$ consisting of infinitely many isomorphic copies of I over disjoint sets of oid's and an ordering β on the elements of $\bar{I}$ such that ω is also a prefix of $enc_\beta(\bar{I})$. Clearly, $\bar{I}$ and β exist and by genericity $q(\bar{I})$ is infinite. This is a contradiction, since q is computable and therefore produces only finite answers. Similarly, there is no nontrivial 0-eventually computable query that always produces finite answers.

Observe that 0-computability makes sense on finite databases (and indeed corresponds to the standard notion of computability). However, we are concerned here with Web instances, which are infinite. Since terminating computation remains important in this context, we modify our notion of computability to allow for meaningful finite computations.

A second attempt

The source of the problem with our definitions so far is that any finite computation on $enc_\alpha(I)$ sees an arbitrary finite sample of I, determined by the encoding. This is unsatisfactory, because we clearly want to allow the possibility of meaningful finite computations. This leads naturally to the solution adopted all along in practice, which is to carry out the computation starting from a designated Web object. This particular object is then part of the input to the query.

This can be formalized as follows. A Web query is a mapping q associating to each Web instance I and object $o \in I(Obj)$, a subset $q(o, I)$ of $I(Obj)$. The object o is called the *source (of the query)*. The definitions of computable and eventually computable query are the same, except that the encoding of the input on the Web machine input tape is now $enc_\alpha(o)\#\#\#enc_\alpha(I)$. *We henceforth adopt the above definitions of Web query, computable query, and eventually computable query.*

Observe that the presence of a source object indirectly allows to refer to more that one "constant" vertex in a query. This can be done by linking the source object to other objects we wish to name, by edges with new labels.

Example 1. The notions of computable and eventually computable queries are illustrated by the following queries on input (o, I):

1. computable:
   - Find the objects reachable from o by a path labeled $a.b.c$ (an a-labeled edge, followed by a b-labeled edge, followed by a c-labeled edge).
   - Find the objects o' such that there is a path of length at most k from o to o'.
   - Find all objects lying on a cycle of length at most 3 which contains o.
2. eventually computable with possibly infinite answers (so not computable):
   - Find the objects reachable from o.
   - Find the objects referencing o.
   - Find the objects belonging to a cycle.
3. eventually computable with finite answers, but not computable:
   - Find the objects on the shortest cycle containing o.
   - Find the object(s) at the shortest distance from o that reference o.
4. not eventually computable:
   - Find all objects that do not belong to a cycle.
   - Find all objects which are not referenced by any other object.
   - Output o iff all objects reachable from o have non-nil references[4].

In particular, it is clear from the above examples that computable and eventually computable properties are not closed under complement.

[4] Nil references can be modeled by references to a special object named *nil*.

Browse and Search

The Web machine captures a very general form of computation on the Web. However, two particular modes of computation on the Web are prevalent in practice: *browsing* and *searching*. We next define two machine models that capture more directly such computation. The first, called a *browser machine*, models browsing. The second, called a *browse/search machine* models browsing and searching combined.

The idea underlying the browser machine is to access the Web navigationally, by following object references starting from the input object o. A browser machine has an infinite browsing tape, an infinite work tape, and a right-infinite output tape. It is equipped with a finite state control which includes a special state called *expand*. The computation of the machine on input (o, I) is as follows. Let α be a fixed successor relation on the elements of I. Initially, the browsing tape contains the encoding $enc_\alpha(o)$ of the source object o. If the *expand* state is reached at any point in the computation and the browsing tape contains the encoding $enc_\alpha(o')$ of some object o' in $I(Obj)$, this is replaced on the browsing tape by $enc_\alpha(\widehat{o'})$ (i.e., the encoding of the finite description of o', see earlier notation for encodings).

A query q is computable by a browser machine if there exists a browser machine which on input (o, I) halts and produces on the output tape the encoding of $q(o, I)$. The definition of query eventually computable by a browser machine is analogous.

Obviously, browser machines have limited computing ability, since they can only access the portion of the Web reachable from the input object. However, this is an intuitively appealing approach for controlling the computation. We next prove a result which confirms the central role of this style of computation in the context of the Web.

Theorem 1. *Every generic and computable Web query is browser computable.*

Proof. (Sketch): Since our formalism is a departure from familiar terrain, we provide some detail in this first proof. Let q be a generic and computable Web query and W a Web machine computing q. Let (o, I) be an input for q. Let I_o denote the subinstance of I consisting of descriptions of all objects reachable from o. If we show that $q(o, I) = q(o, I_o)$ we are done, since $q(o, I_o)$ is clearly computable by a browser machine. There is however one difficulty: I_o may be finite, in which case it is not a Web instance.

To fix this problem, let $\bar{I}_o$ be I_o augmented with an infinite set *New* of new objects with the same new value, say 0, and without references. Now $\bar{I}_o$ is surely a Web instance. For technical reasons, we also need to similarly augment I. Let $\bar{I}$ be I augmented with the objects in *New*. We will show that:

1. $q(o, I) = q(o, \bar{I})$, and
2. $q(o, \bar{I}) = q(o, \bar{I}_o)$.

For suppose that (1) and (2) hold. Then, $q(o, I) = q(o, \bar{I}_o)$. Now a browser machine W' can compute $q(o, I)$ by simulating the computation of $q(o, \bar{I}_o)$. The browser generates on a portion of its work tape an encoding of a prefix of (o, I_o), and starts simulating W on this input tape. Whenever W attempts to move past the right end of the input tape, W' extends the tape by either browsing I_o or (if I_o is finite and has already been exhausted), by generating the encoding of an object in *New*. Since objects in *New* are standard, their encodings can simply be made up by the browser.

To prove (1), let α be a successor relation on all elements of I. The computation of W on $enc_\alpha(o, I)$ halts after W has inspected a finite prefix ω of $enc_\alpha(o, I)$. Clearly, there exists a successor relation β on the elements of $\bar{I}$ such that ω is also a prefix of $enc_\beta(o, \bar{I})$. Thus, $q(o, I) = q(o, \bar{I})$.

The proof of (2) is similar, starting from the computation of W on input $(o, \bar{I}_o)$.

Remark. (i) Observe that the previous result does not hold without the assumption that Web instances are infinite. Consider the following query: on input (o, I), output o_1, o_2 if I consists precisely of o and two other objects o_1, o_2 pointing to o. This would be computable by a Web machine but not by a browser machine. (ii) In addition to computable queries, browser machines can also compute queries that are eventually computable but not computable (e.g., "Find all objects reachable from o"). However, there exist eventually computable queries which are not eventually computable by a browser machine, such as "Find all objects in I".

We next augment browser machines with a search mechanism. The search is essentially a selection operation on a relation in the schema, whose condition specifies a conjunction of a finite set of (in)equalities involving an attribute and a constant. Examples of selections are: (i) $\sigma_{value=SGML}(Val)$ that returns all tuples $Val(o,SGML)$ where o is an object whose value is "SGML"; (ii) $\sigma_{label=Department}(Ref)$ that selects all edges with label "Department"; (iii) $\sigma_{label=A \wedge destination=556}(Ref)$ that returns all edges with label A and oid 556 as destination; and (iv) $\sigma_{source=source}(Ref)$ that returns all edges. In general, a search triggers an eventually computable subquery, whose result may be infinite. This leads to the problem of integrating nonterminating subcomputations into the computation of a query. We adopt the following model.

A *browse/search machine* is a browser machine augmented with a right-infinite search-answer tape and a separate search-condition tape. There is a distinguished *search* state. The computation of the machine is nondeterministic. A search is triggered by writing a selection operation on the search-condition tape, then entering the search state. The search-answer tape functions similarly to the answer tape of an eventually computable query. Answers to previously triggered searches arrive on the search-answer tape at arbitrary times and in arbitrary order. More precisely, suppose the set of selections triggered up to some given point in the computation is $\{\sigma_1, \ldots, \sigma_n\}$. In any subsequent move of the machine, a (possibly empty) finite subset of the answers to some of the σ_i's is appended to the search-answer tape. This is non-deterministic. The order in which answers are produced is arbitrary. Each tuple in the answer to σ_i is prefixed by σ_i (everything is encoded in the obvious way). It is guaranteed that all answers to a triggered search will be eventually produced, if the computation does not terminate. However, note that there is generally no way to know at a given time if all answers to a particular search have been obtained.

The rest of the computation occurs as in the browser machine. A Web query q is computable by a browse/search machine if there exists a browse/search machine W such that *each* computation of W on input (o, I) halts and produces an encoding of $q(o, I)$ on the answer tape[5]. The definition of query eventually computable by a browse/search machine is analogous.

What is the power of browse/search machines? This is elucidated by the following result.

Theorem 2. *(i) A generic Web query is eventually computable iff it is eventually computable by a browse/search machine.*
(ii) A generic Web query is computable iff it is computable by a browse/search machine.

Proof. (Sketch): For (i), consider first a query that is eventually computable by a browse/search machine M. The *expand* operations of M are easy to simulate in finite time by a Web machine (but note that this uses the fact that the encodings of tuples describing a given object are clustered together). Searches are simulated as follows. For each selection, the Web machine scans the input tape from left to right in search of answers to the selection. When a tuple in the answer is found, its encoding is placed on a portion of the worktape that simulates the search-answer tape of M. The searches (which never terminate) are interleaved with the rest of the simulation in some standard way.

Conversely, suppose q is eventually computed by a Web machine W. A browse/search machine can simulate W as follows. First, it triggers a search on a selection condition true of all objects. As objects arrive on the search-answer tape, they are expanded and encoded on the work tape using *expand*. This is interleaved with a simulation of W on the portion of the input tape constructed so far.

Part (ii) follows immediately from Theorem 1.

3 Query Languages

It is tempting to use classical declarative query languages in the context of the Web. However, it is not clear to what extent such languages are appropriate in this framework. We examine this issue in light of the notions of (eventual) computability discussed so far. Specifically, we consider the languages FO (first-order logic), Datalog, and Datalog$^\neg$. Due to space limitations, we assume familiarity with the above languages (e.g., see definitions in [AHV94]). All programs we

[5] However, it should be clear that a browse/search machine that uses the search feature in a nontrivial way cannot terminate.

consider here use as input the Web relations *Obj, Ref* and *Val*, as well as one constant *source* that is interpreted as the object o in an input instance (o, I).

For each language, we are interested in the following questions:

(i) are the queries in the language (eventually) computable?
(ii) which fragment of each language can be implemented by browsers?

As it turns out, conventional wisdom cannot be counted upon in this context. To begin with, FO is no longer a nice, tractable language: it expresses queries that are not eventually computable. We will see that negation is the main source of problems in the languages we consider. This is not surprising, given that neither the (eventually) computable queries nor the queries computable by browser machines are closed under complement. We therefore begin our discussion with languages without negation: positive FO (FO without negation or universal quantification, denoted FO^+) and Datalog. The following is easily shown.

Theorem 3. *All FO^+ and Datalog queries are eventually computable.*

In particular, every FO^+ and Datalog query can be implemented by a browse/search machine. Clearly, the fragments implementable by a browser machine are of special interest. Note that navigational languages proposed for the Web are implementable by browsers. In particular, the languages based on specification of paths from the source object using regular expressions (e.g., see [AMM96]), are fragments of Datalog implementable by browsers. We isolate fragments of Datalog and FO^+ (eventually) computable by browsers by a syntactic restriction on variables which limits their range to values reachable from the source. We provide the definition for Datalog (this induces an analogous restriction on FO^+, since this can be viewed as nonrecursive Datalog).

Definition 4. The set of *source-range-restricted* variables in a Datalog rule r is the minimum set of variables in r satisfying:

- if $R(\mathbf{u})$ occurs in the body of the rule, R is some idb predicate and x is one of the variables of $\mathbf{u}$, then x is source-range-restricted;
- if x is the source constant or x is source-range-restricted and $Ref(x, y, z)$ occurs in the body of the rule, then y, z are source-range-restricted; and
- if x is the source constant or x is source-range-restricted and $Val(x, y)$ occurs in the body of the rule, then y is source-range-restricted.

A rule is *source-safe* (ss) if all its variables are source-range-restricted. A program is *source-safe* if all its rules are source-safe.

For example, the first Datalog program below is source-safe, and it is eventually computable by a browser machine. The second program is not source-safe. It is eventually computable, but not by a browser machine alone.

reachable nodes	$answer(source) \leftarrow$	
	$answer(t')$	$\leftarrow answer(t), Ref(t, x, t')$
nodes leading	$answer(source) \leftarrow$	
to the source	$answer(t)$	$\leftarrow answer(t'), Ref(t, x, t')$

We can now show the following.

Theorem i *All ss-FO^+ queries are computable by a browser machine.*
(ii) All ss-Datalog queries are eventually computable by a browser machine.

We next consider languages with negation. As expected, things become more complicated. Even without recursion, one can easily express queries which are not eventually computable. Consider the FO query

$$\{x \mid Ref(source, A, x) \wedge \neg\exists y(y \neq source \wedge Ref(y, A, x))\}.$$

This asks for the (finite set of) objects x which are referenced with an edge labeled A by *source* and by no other object. It is easy to see that this query is not eventually computable.

Besides FO, we will consider Datalog$^\neg$ with stratified, well-founded, and inflationary semantics. To obtain fragments eventually computable by browser machines, it is natural to extend the source-safe restriction to these languages. The definition of ss-Datalog$^\neg$ is precisely the same as for Datalog, with the proviso that all occurrences of predicates required by the definition to ensure source-range-restriction must be *positive* occurrences. (More precisely, the definition is obtained by replacing "occurs" by "occurs positively" in the definition of source-safe for datalog.) A definition of source-safe FO program in the same spirit can be given (we omit the details). It is straightforward to show:

Theorem 5. *All queries in ss-FO are computable by a browser machine.*

Consider now ss-Datalog$^\neg$. This language provides some interesting surprises. The classical stratified and well-founded semantics do not appear to be well-suited to express eventually computable queries, whereas inflationary semantics is quite well-behaved. First, recall that in the finite case (i) FO is subsumed by stratified-Datalog$^\neg$ which is subsumed by Datalog$^\neg$ with well-founded semantics [GRS88], and (ii) Datalog$^\neg$ with well-founded semantics (with answers reduced to their positive portion[6]) is equivalent to Datalog$^\neg$ with inflationary semantics [Gel89]. In the infinite case, things are different: (i) continues to hold but (ii) does not.

It is quite easy to see that ss-Datalog$^\neg$ with stratified semantics expresses queries that are not eventually computable. For example, consider the stratified ss-Datalog$^\neg$ program:

$$\begin{array}{llll} R(source) \leftarrow & & R(t') & \leftarrow R(t), Ref(t, x, t') \\ R_1(t') & \leftarrow R(t), R(t'), Ref(t, A, t') & answer(t) & \leftarrow R(t), \neg R_1(t) \end{array}$$

The query asks for all vertices reachable from the source, without in-going edges labeled "A" from any other vertex reachable from the source. One can show that the stratified semantics (so also the well-founded semantics) of this query is not eventually computable.

For inflationary semantics, we are able to show:

[6] Recall that well-founded semantics uses a 3-valued model.

Theorem 6. *Every query in ss-Datalog$^\neg$ with inflationary semantics is eventually computable by a browser machine.*

Generally, there are queries which are eventually computable (and even computable) by a browser machine, which are not expressible in ss-Datalog$^\neg$ with inflationary semantics. An example of such a computable query is "Output o iff there is an even number of objects x such that $Ref(o, A, x)$." This is a familiar difficulty in the theory of query languages, and is due to the lack of an order on the domain. Let us consider Web instances augmented with a total order relation on all oid's in *Obj*. Call such a Web instance *ordered*. Also, a Web instance (o, I) is *source-infinite* if there are infinitely many objects reachable from o. Otherwise, the instance is called *source-finite*. We can show the following:

Theorem 7. *(i) The language ss-Datalog$^\neg$ with inflationary semantics expresses exactly the queries eventually computable by a browser machine on ordered, source-infinite Web instances.*
(ii) The language ss-Datalog$^\neg$ with inflationary semantics expresses exactly the queries computable by a browser machine in polynomial time (with respect to the number of objects reachable from source*) on ordered, source-finite Web instances.*

The proof involves a simulation of browser machines. The tape cells of the machine are encoded using indexes consisting of objects reachable from the source. Recall that browser machines may not terminate, so infinitely many cells may have to be encoded. This only works for source-infinite instances (even in the case when the browser machine itself only inspects finitely many objects reachable from the source). On source-finite instances ss-Datalog$^\neg$ with inflationary semantics can only construct polynomially many indexes for tape cells.

Theorem 7 allows to show an interesting connection between the ss-Datalog$^\neg$ languages with various semantics for negation.

Proposition 8. *On ordered Web instances, every query eventually computable by a browser machine that is expressible in ss-Datalog$^\neg$ with well-founded semantics is also expressible in ss-Datalog$^\neg$ with inflationary semantics.*

The proof uses the fact that on source-finite Web instances every ss-Datalog$^\neg$ program with well-founded semantics can be evaluated in time polynomial in the number of objects reachable from the source. Thus, Proposition 8 follows from a complexity/completeness argument rather than from an explicit simulation. It remains open to find a uniform simulation of ss-Datalog$^\neg$ with well-founded semantics which are eventually computable by browsers, by ss-Datalog$^\neg$ with inflationary semantics. It also remains open whether Proposition 8 holds for unordered Web instances.

In view of these results, ss-Datalog$^\neg$ with inflationary semantics emerges as a particularly appealing language in the context of the Web.

Remark. The notion of source-safety was developed to ensure that programs can be implemented by a browser. One could develop a less restrictive notion of safety geared towards eventual computability, which would guarantee that the program can be implemented by browsing and searching combined. Consider for instance Datalog$^{(\neg)}$. Recall that Datalog queries (without negation) are eventually computable with browse and search, while ss-Datalog$^{\neg}$ programs with inflationary semantics are eventually computable with browsers alone. One could relax the source-safety restriction of ss-Datalog$^{\neg}$ by allowing a mix of idb's defined by positive rules and idb's defined by source-safe rules. Hybrid rules that are neither positive nor source-safe are allowed if idb's occurring negatively are defined only by source-safe rules and variable occurring under negation are also bound to positive occurrences of some predicate. Such programs express (with inflationary semantics) queries eventually computable by browse and search. We omit the details here.

4 Conclusions

We explored some basic aspects of querying and computing on the Web. In doing so, we revisited and adapted fundamental concepts from the theory of database query languages, such as genericity and computability. There are substantial differences, arising from the fact that we model the Web as a semistructured, infinite object. Some of our results can be viewed as *a posteriori* formal justification for much of the computation style adopted in practice in the context of the Web, based on browsing from a given source object.

We considered FO, Datalog and Datalog$^{\neg}$ in the context of the Web, and characterized them with respect to (eventual) computability. We also identified fragments in each language implementable by browsing alone. There were some surprises: FO is no longer the nicely behaved language we are used to from the finite case. And among semantics for negation in Datalog$^{\neg}$, stratified and well-founded semantics have fundamental shortcomings, whereas inflationary semantics emerges as particularly appealing in this context. Although it is unlikely that FO, Datalog, or Datalog$^{\neg}$ will be used as such to query the Web, the results can guide the design of more practical languages. In particular, we believe that the nice properties of source-safe Datalog$^{\neg}$ with inflationary semantics suggest useful ways to extend previously proposed languages based on browsing.

As emphasized in the introduction, our abstraction of the Web left out important aspects which we plan to include in future investigations. Perhaps the most important are the communication costs associated with browsing and searching, and the notion of locality. Locality could be introduced in our model by having two-sorted edges in the reference graph *Ref*: local and remote, with the added condition that each connected component of the subgraph of *Ref* consisting of local edges is finite. The fact that local browsing/searching is guaranteed to terminate can in turn be exploited at the language level by allowing explicit reference to local links in the language. Locality is indeed an explicit notion in some languages proposed for the Web [AMM96]. It is natural then to provide extended notions of safety based on locality of browsing and searching.

References

[A97] S. Abiteboul. Querying semistructured data. In *Proc. ICDT*, Greece, 1997.

[AHV94] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading-Massachusetts, 1994.

[AMM96] G. A. Mihaila A. Mendelzohn and T. Milo. Querying the world wide web, 1996. draft, available by ftp: milo@math.tau.ac.il.

[AV88] S. Abiteboul and V. Vianu. Procedural and declarative database update languages. In *Proc. ACM Symp. on Principles of Database Systems*, pages 240–250, 1988.

[BDS95] P. Buneman, S. Davidson, and D. Suciu. Programming constructs for unstructured data. In *Proc. DBPL*, 1995.

[BLCL+94] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Nielsen, and A. Secret. The World-Wide Web. *Comm. of the ACM*, 37(8):76–82, aug 1994.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *SIGMOD'94*, ACM, 1994.

[CH80] A.K. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.

[CH93] Completeness Results for Recursive Data Bases. In *Proc. 12th ACM Symp. on Principles of Database Systems*, pages 244–252, 1993.

[CM89] Mariano P. Consens and Alberto O. Mendelzon. Expressing structural hypertext queries in graphlog. In *Proc. 2nd. ACM Conference on Hypertext*, pages 269–292, Pittsburgh, November 1989.

[Gel89] A. Van Gelder. The alternating fixpoint of logic programs with negation. In *Proc. ACM Symp. on Principles of Database Systems*, pages 1–11, 1989.

[GRS88] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. In *Proc. ACM Symp. on Principles of Database Systems*, pages 221–230, 1988.

[GZC89] Ralf Hartmut Güting, Roberto Zicari, and David M. Choy. An algebra for structured office documents. *ACM TOIS*, 7(2):123–157, 1989.

[KKR90] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. In *Proc. 9th ACM Symp. on Principles of Database Systems*, pages 299–313, 1990.

[KP88] P. G. Kolaitis and C.H. Papadimitriou. Why not negation by fixpoint? In *Proc. ACM Symp. on Principles of Database Systems*, pages 231–239, 1988.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In *Proc. of VLDB'95*, pages 54–65, 1995.

[LSS96] 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.

[MW93] T. Minohara and R. Watanabe. Queries on structure in hypertext. In *Foundations of Data Organization and Algorithms, FODO '93*, pages 394–411. Springer, 1993.

[MW95] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comp.*, 24(6), 1995.

[Q+95] D. Quass et al. Querying semistructured heterogeneous information, 1995.

This article was processed using the LaTeX macro package with LLNCS style