5th Intl. Conf. on Deductive and Object-Oriented Databases (DOOD'97), December 8–12, 1997, Montreux, Switzerland. LNCS 1341, Springer.

On a Declarative Semantics for Web Queries

Rainer Himmeröder* Georg Lausen
Bertram Ludäscher Christian Schlepphorst*

Institut für Informatik, Universität Freiburg, Germany
{himmeroe,lausen,ludaesch,schlepph}@informatik.uni-freiburg.de

Abstract. With the increasing importance of the World Wide Web as an information source, there is a growing interest for integration of Web and database technology. Several Web query languages have been presented to overcome the shortcomings of conventional search engines, most of them lacking a formal semantics. In this paper, we propose F-logic as a means to explore, query and restructure Web information, and to integrate it with a local database. Two major advantages of this approach are its declarative semantics, based on the semantics of F-logic, and the rich modeling capabilities which make F-logic particularly suitable to handle heterogeneous data. The presented semantics directly yields a bottom-up evaluation algorithm that can be easily incorporated into the existing F-logic prototype FLORID.

1 Introduction

With the World Wide Web (WWW) an information repository is provided on top of the Internet which is interesting and relevant for science, research and industry as well. However, it can be hard to locate documents of interest in this huge and almost unstructured system. In practice, the only alternative to browsing the network by following links is the use of search engines, e.g. Alta Vista or HotBot, which return lists of documents containing the given keywords. The limitations of browsing as a search technique are obvious, given the large number of documents in the Web. Search engines lack the possibility to take the *structure* of the Web into consideration. Not surprisingly, there has been a very strong interest in providing query languages for the Web similar to those developed for database systems, see e.g. [KS95, LSS96, MMM96, Suc97].

The benefits of such query capabilities are locating, filtering and presenting WWW-held information in a declarative way. In this paper, we use F-logic [KLW95] for this purpose. F-logic is well-suited for querying and restructuring Web data, since it combines a logical semantics with the powerful modeling capabilities of the object-oriented paradigm. The querying facilities take document structure and Web topology into account. Here, restructuring means to reorganize parts of the Web into a database view. The main features of our approach are:

* The work of these authors is supported by the Deutsche Forschungsgemeinschaft, La 598/3-2.

- a declarative semantics for Web queries is defined by incorporating Web access into an existing DOOD framework. This allows to integrate Web data with a local database.
- explored and unexplored parts of the Web can be distinguished. Exploration of the Web corresponds to extending the (partial) F-logic model by newly derived information. Therefore, the access semantics can be smoothly integrated into the existing bottom-up evaluation technique.

After giving a brief survey of F-logic in Section 2, we present in Section 3 our model of the Web. In Section 4, we show by example how to query and restructure the Web using our extension of F-logic. The underlying semantics of Web exploration and the resulting bottom-up evaluation algorithm are given in Section 5. We conclude with a discussion about further extensions of our proposal, which has been implemented in the F-logic prototype FLORID[1] [FHK+97].

Related Work At present, there are particularly two interesting streams of publications discussing the integration of the Web and database technology. The first one are the so-called *Web query languages* that provide features for querying contents and structure of the Web analogously to known database query languages. WebSQL [MMM96] has a formal semantics in terms of a virtual graph model, and a notion of locality for query analysis. Regular path expressions enable querying the topology of the Web. In contrast, W3QL [KS95] is a SQL-like Web language without formal semantics, but has additional features as filling forms. Lakshmanan et al. [LSS96] propose WebLog, a rule language for querying and restructuring the Web. Instead of regular expressions as used in WebSQL, WebLog relies on possibly recursive rules. The syntax resembles that of F-logic, but no formal semantics for WebLog is given. Abiteboul and Vianu [AV97] introduce theoretical foundations to investigate the computability of Web queries, based on so-called *Web machines*, but do not present a concrete Web query language. Mendelzon and Milo [MM97] present a formal model of Web queries which is closely related to [AV97], in which especially the effects of limited access to data and the lack of concurrency are discussed. In contrast to the approach of Abiteboul and Vianu, they consider the Web as a *finite* structure.

The other direction is centered around the topic of *semi-structured data* [Abi97]. In the context of data integration, especially integration of data found on the Web, this topic has received a lot of attention [Suc97]. The term "semi-structured data" refers to data that has a certain structure, but is not as strictly structured (typed) as data in conventional database systems. Most information in the Web is present in the form of Hypertext Markup Language (HTML) documents, a typical example for semi-structured data. Two prominent query languages for semi-structured data are Lorel [AQM+] and UnQL [BDHS96]. Both use a data model that is based on labeled graphs. Lorel is an extension of OQL [Cat94], providing additional features as *coercion* and *general path expressions*

[1] FLORID is available from
`http://www.informatik.uni-freiburg.de/~dbis/flogic-project.html`

that are useful for querying semi-structured data. Coercion means to relax the typing requirements of OQL by extending comparison predicates and base functions. Thus, the user does not have to know the precise types of objects. The idea of general path expressions is to allow queries even if the structure is not completely known. An important feature of UnQL is a construct called *traverse* that allows tree restructuring up to an arbitrary depth. UnQL can be translated into UnCal, a calculus enabling certain optimization techniques.

Our approach follows the first direction by enhancing F-logic's semantics to cope with Web queries. However, F-logic also has powerful capabilities to handle data from heterogeneous sources, an important problem of semi-structured data. Indeed, F-logic matches many of the criteria for query languages tailored for semi-structured data given in [Abi97].

2 Preliminaries: F-Logic in a Nutshell

In this section, we briefly review the basic constructs of F-logic and its extension by path expressions. For a comprehensive description of the syntax and semantics the reader is referred to [KLW95, FHKS97b]. In F-logic, objects have logical oids, so-called *id-terms* which are ground first-order terms built from a set of functors $\mathcal{F}$. For example, john and father(mary) denote objects holding information about individuals. Facts about objects are syntactically represented by *molecules*, e.g.:

person[name $\Rightarrow$string; children $\Rightarrow\!\!>$person].
john:employee[name $\rightarrow$"John Smith"; children $\rightarrow\!\!>$\{mary,bob\}].
father(mary)[address@(1997)$\rightarrow$"Main St. USA"; spouse $\rightarrow$sally].

The first molecule gives the types of the methods name und children on a class person. The second molecule states, that john is a employee and gives the values of these methods. The arrow "$\rightarrow$" denotes single-valued methods whereas "$\rightarrow\!\!>$" denotes multi-valued methods[2]. Signatures of methods are specified by the double-shafted arrows "$\Rightarrow$" and "$\Rightarrow\!\!>$". 0-ary methods are also called *attributes*. Note that F-logic does not distinguish between method names and other logical oids, i.e., methods are objects as well.

Molecules may be split into several *atoms* where each atom expresses exactly one property. The molecule about john is equivalent to the conjunction of atoms

john:person, john[name$\rightarrow$"John Smith"],
john[children$\rightarrow\!\!>$\{mary\}], john[children$\rightarrow\!\!>$\{bob\}].

Objects that share the same properties are grouped into classes. For example john : employee means that john is an *instance* of *class* employee and employee :: person means that employee is a *subclass* of person. Due to the closure properties john is also an instance of class person. F-logic overcomes the strict separation of schema manipulation and access to data and allows uniform handling of data

[2] F-logic distiguishes between inheritable and non-inheritable properties. We do not consider inheritance here, since it is not needed for our purpose.

on the instance and on the schema level. In F-logic, a higher-order syntax is obtained by allowing variables also at method and class positions. So objects may act at the same time as classes, objects, and methods. As usual, in F-logic variables are capitalized, while constants are denoted by lower case letters.

Path Expressions. In [FLU94] an extension of F-logic by path expressions is introduced. The idea of path expressions is to follow a link between objects without having to write down explicit join conditions. Path expressions can be used to reduce the number of variables in a program and often result in more concise and readable programs. Consider for example the rules:

$$\mathsf{X[grandfather} \rightarrow \mathsf{Z]} \leftarrow \mathsf{X[father} \rightarrow \mathsf{Y]}, \mathsf{Y[father} \rightarrow \mathsf{Z]}.$$
$$\mathsf{X[grandfather} \rightarrow \mathsf{Z]} \leftarrow \mathsf{X.father[father} \rightarrow \mathsf{Z]}.$$

In the second rule X.father is a path expression denoting the result of the single-valued method father applied to any object that defines this method. Multi-valued paths are denoted by "..". Paths and molecules may be mutually nested:[3]

$$\mathsf{X[oldgrandma} \rightarrow \mathsf{Z]} \leftarrow \mathsf{X.father[age} \rightarrow \mathsf{A].father[spouse} \rightarrow \mathsf{Z]}, \mathsf{A} > 60.$$

Paths and molecules together form the set of *references*. Besides enabling efficient object navigation, single-valued path expressions, when appearing in the head of a rule, can also create objects[4]. This effect occurs whenever a path expression consists of a host object with a method application that has not been defined otherwise:

$$\mathsf{X.father} : \mathsf{man} \leftarrow \mathsf{X} : \mathsf{man}, \mathsf{not}\ \mathsf{X} \doteq \mathsf{adam}. \tag{2.1}$$

If a man other than adam is known, e.g. john, a new object john.father representing his father is created (if the method father was not already defined for this person) and made an instance of man, too. Note, that this rule will then create an infinite number of ancestors john.father.father, john.father.father.father etc. This shows that object creation has to be used with care, similar to function symbols, to ensure the existence of a finite model.

3 The Web Model

Every resource available in the Web has a unique address, called *Uniform Resource Locator* (URL). HTML documents may contain *hyperlinks* to other documents, that is, URLs with labels. Therefore it seems quite natural to have an abstract view of the Web as a labeled directed graph where the nodes represent documents and the labeled edges are provided by the hyperlinks (cf. e.g. [MM97]). Due to its size and often low transfer rates, it is practically impossible

[3] The rule expresses that if the father X.father of X is older than 60, then X.father.father's spouse is X's (old) grandmother.

[4] Multi-valued path expressions are not allowed in rule heads as this would lead to semantical problems.

to have access to the Web as a whole. As [MMM96] states, a notion of locality and a selection of interesting parts of the Web is necessary for practical reasons. Therefore, we propose a different, more practically oriented Web model, where only a small part has been transferred to an abstract form and the rest can be accessed by black box functions if needed. Thus, in our model it is not necessary to decide whether the whole Web should be considered as finite, but undetermined as in [MMM96] and [MM97], or as infinite as in [AV97]. Figure 1 shows the exploration cycle between the classes url (for URLs) and webdoc (for Web documents).

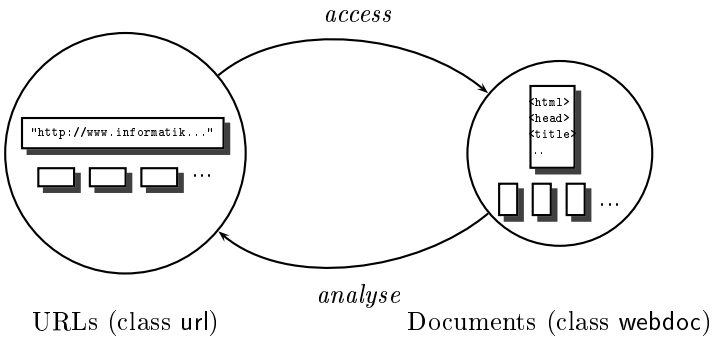

Fig. 1. Exploring the Web by iteration of *access* and *analyse*

The *access* function loads the document associated with an URL u from the Web (this is standard browser functionality), and *analyse* extracts its structure and maps it into database facts. In particular, *analyse* may yield a set of new, previously unexplored URLs. A bottom line for the structure to recognize is given by the following signature:[5]

```
webdoc[self ⇒url; author ⇒string; modif ⇒string;
       type ⇒string; hrefs@(string) ⇒⇒url; error ⇒⇒string].
```

Class webdoc is the base class of all kinds of documents that appear in the Web and provides the properties all documents have in common. In the schema above, self denotes the URL, author the owner, modif the time of last modification, and type the type of the document. If the Web access fails, error returns the reason of the failure (e.g. *server does not exists*, *page not found*, *connection timed out*). Method hrefs, parametrized with the link labels, returns the links in the document.[6] The result type of hrefs is the class url. Furthermore, we can

[5] This does not mean that all attributes are mandatory. If for example an error occurs, every other attribute will be void.

[6] A more detailed modeling could also include the offset of the link as a parameter, cf. [MM97].

declare subclasses of Web documents:

```
htmldoc::webdoc[title ⇒string; text ⇒string].
```

specifies that HTML documents are Web documents with the additional attributes title and text containing the title and the ASCII representation of the document. Clearly this schema does not reflect the full structural information covered by HTML documents but exemplifies the possibilities. It may be refined to greater detail for practical use. Deriving further subclasses of webdoc can be done to handle special features of other document types available on the Web, e.g. BibTeX files, tables etc.

4 Exploring the Web with F-Logic

The problem when integrating the exploration cycle into the F-logic framework lies in the necessity to load a beforehand unknown number of Web documents and add their data to the model. Loading has to be completely data-driven; the user must have the possibility to explore new documents depending on information and links found in already known documents. Hence we have to create new oids for those documents and feed in the result of the *analyse* function. To solve this problem, we exploit the object creation feature of path expressions, thus integrating the Web access without any syntactic patches. All we have to do is to extend the semantics for the reserved method get (used to explore the Web) on the class of URLs:

```
url::string[get ⇒webdoc].
```

If the method get is defined for an instance u of class url, we force u.get to be an instance of webdoc and have the respective methods defined. Thus, when this method definition is derived during the evaluation of a program, we have to call the function

$$explore(u) := analyse \circ access(u)$$

and add the delivered facts to the partial model. This extended semantics is formally exposed in the next section.

The following three facts define the Web address of our institute's homepage, and fetch its contents:

```
ourHomepage ≐ "http://WWW.Informatik.Uni-Freiburg.DE/".
ourHomepage : url.
ourHomepage.get[].
```

Here, the equality predicate is used to define a short name for the given URL. When we ask about the name of the resulting document, we obtain the path expression as an answer:

```
?– X[self → ourHomepage].
X = "http://WWW.Informatik.Uni-Freiburg.DE/".get
```

The following examples demonstrate how the contents and structure of interesting parts of the Web can be explored using this access mechanism. We assume a set of built-in string manipulation predicates, like substr(a,b), which is true if a is a substring of b. Clearly, more elaborated functions e.g. to match regular expressions can be provided.

Example 1 First show the author and the time of last modification of our institute's homepage. Then show all links that are labeled with "database".

```
?– ourHomepage.get[author →X; modif →Y].
?– ourHomepage.get[hrefs@(L)↠X], substr("database",L).
```
□

For the following examples, assume that a publishing company has collected the base URLs of all computer science departements. These URLs form the instances of the subclass csurl of url.

Example 2 Find all HTML pages belonging to the computer science departements.

```
csurl::url.
X.get:cspage  ⇐ X:csurl.
Y.get:cspage  ⇐ X:cspage[type →"html"; hrefs@(L) ↠Y],
                  substr(Z:csurl,X.self).
```
□

This example makes use of a recursive rule to locally traverse the graph structure of the Web. To ensure that only documents of the given cs departements are visited, the last subgoal restricts the followed links to those starting with a csurl. Without any restriction, all documents reachable from cs homepages would be loaded, possibly a major part of the whole Web. If no other precautions are taken (such as limiting the depth of the graph traversal), the user is responsible to specify which parts of the Web are interesting for her.

As mentioned before, this "relevant" part of the Web may be integrated into an F-logic database thereby allowing to integrate Web queries with the local database.

Assume that the publishing company stores the names of their customers as instances of class customer in the database. After getting all interesting Web pages in Example 2, they want to know which Web page authors are not customers of their products yet.

Example 3 Annotate the cs pages whose authors are customers. Query those pages whose authors are not customers.

```
Y[customerpage →true]  ⇐ Y:cspage[author →(X:customer).name].
?– Y:cspage, not Y[customerpage→true].
```
□

The last two examples demonstrate how explored parts of the Web can be reorganized and reused. Typically this form of restructuring is either done by grouping documents into classes as in Example 2, or by adding new properties to them as in Example 3.

In the next example, contents and structure of the part of the Web which was explored in Example 2 are queried:

Example 4 Find all pairs of cs pages on different servers where the string "database" occurs in the title and which reference each other directly.

```
related(X,Y) ←
    X:cspage[title→T1; hrefs@(L1)↠Y.get[title→T2; hrefs@(L2)↠X.self]],
    substr("database",T1), substr("database",T2), not sameServer(X,Y).
```

Note, that Y.get only matches those documents that are already explored. New documents are accessed only when ".get" appears in rule *heads*. □

5 Semantics of Web Access

The semantics of extending F-logic by path expressions was given in [FLU94] in terms of a semantic structure

$$I = (U, \in_U, I_{\mathcal{N}}, I_{\rightarrow}, I_{\twoheadrightarrow}),$$

where U is the universe of oids, $\in_U$ the class membership relation, $I_{\mathcal{N}}$ a function mapping a set $\mathcal{N}$ of external names onto oids and $I_{\rightarrow}, I_{\twoheadrightarrow}$ interprete single- and multi-valued methods. For this exposition, we want to use Herbrand interpretations instead, since this makes the formal treatment of minimal models and deduction operators easier. Herbrand interpretations can be mapped to I-structures as described in [KLW95], Section 9. Here, we disregard inheritance, typing and negation. They can be addressed as in [KLW95].

5.1 Herbrand Models

Given a (finite) set of object constructors $\mathcal{F}$ (i.e. functors of arbitrary arity), the set of all ground id-terms is denoted by $U(\mathcal{F})$. In presence of path expressions, $U(\mathcal{F})$ is not sufficient as Herbrand universe, because single-valued path expressions may create new objects and therefore new logical oids. Thus, we need to include all path expressions relevant for object creation in our universe. For example if thisdoc:webdoc[author →"john"] is true, the path expression thisdoc.author has to be added to the Herbrand universe. In other words, we have to close $U(\mathcal{F})$ wrt. single-valued path constructors; the resulting closure is denoted by $\overline{U(\mathcal{F})}$.[7]

Definition 1 (Herbrand Universe, Herbrand Base) The *(extended) Herbrand universe* $\overline{U(\mathcal{F})}$ is the smallest set such that:

- $U(\mathcal{F}) \subset \overline{U(\mathcal{F})}$,
- if $t_0 \in \overline{U(\mathcal{F})}$, then $(t_0) \in \overline{U(\mathcal{F})}$,

[7] Definitions that have been extended by path expressions in comparison to [KLW95] are overlined.

- if $m, t_0 \in \overline{U(\mathcal{F})}$, then $t_0.m \in \overline{U(\mathcal{F})}$,
- if $m, t_0, \ldots, t_n \in \overline{U(\mathcal{F})}$, then $t_0.m@(t_1, \ldots, t_n) \in \overline{U(\mathcal{F})}$.

This is a subset of the set of all references (i.e. of the union of all paths and molecules). Elements of $\overline{U(\mathcal{F})}$ are called *pure references*. Note that the second condition is necessary to distinguish between path expressions with different execution order. By default, path expressions are evaluated from left to right. So, $a.b.c$ means to apply the method b to the object a and then to call c on the result. In contrast, $a.(b.c)$ first calls c on b and then sends the outcome as a method to a.

The *(extended) Herbrand base* $\overline{\mathcal{HB}(\mathcal{F}, \mathcal{P})}$ to $\mathcal{F}$ and a (finite) set of predicates $\mathcal{P}$ (containing $\doteq$) is now defined as the set of all ground atoms (references and predicate atoms) built from elements of $\overline{U(\mathcal{F})}$. □

Herbrand interpretations are subsets of the Herbrand base, i.e., those which define all true atoms. As $\overline{\mathcal{HB}(\mathcal{F}, \mathcal{P})}$ contains all references, they have a truth value as well. A true pure reference $u \in \overline{U(\mathcal{F})}$ is called an *active name*. Since we are not interested in arbitrary subsets of the Herbrand base, we only consider interpretations which satisfy certain natural closure properties:

Definition 2 (Herbrand Interpretation) A subset $H \subset \overline{\mathcal{HB}(\mathcal{F}, \mathcal{P})}$ of the Herbrand base is a *Herbrand interpretation* if it satisfies the following axioms:

- The closure axioms of F-logic (see [KLW95], Section 7), restricted to active names, especially
  - reflexivity, transitivity and acyclicity of subclass relationship,
  - subclass inclusion,
  - reflexivity, transitivity and symmetry of $\doteq$,
  - equal objects have the same properties,
  - functionality constraint for scalar methods.
- The path expression semantics:
  - if $t_0.m \in H$ then $t_0[m \rightarrow t_0.m] \in H$,
  - if $t_0[m \rightarrow t_r] \in H$ then $t_0.m \doteq t_r \in H$,
  - if $t_0.m@(t_1, \ldots, t_n) \in H$ then $t_0[m@(t_1, \ldots, t_n) \rightarrow t_0.m@(t_1, \ldots, t_n)] \in H$,
  - if $t_0[m@(t_1, \ldots, t_n) \rightarrow t_r] \in H$ then $t_0.m@(t_1, \ldots, t_n) \doteq t_r \in H$.
- The *active name axiom*:
  - if $t \in H$ and t contains an element $u \in \overline{U(\mathcal{F})}$, then $u \in H$. □

The last axiom allows a simple declaration of the set of active names in a Herbrand interpretation, namely $U_H := \overline{U(\mathcal{F})} \cap H$. As we are especially interested in finite interpretations, we note that U_H is finite iff H is finite, because there are only a finite number of ways to build ground atoms. Note also, that in a finite interpretation, no method may be totally defined because for a total method meth all the path expressions meth, meth.meth, meth.meth.meth etc. have to be active.

Definition 3 The truth value of formulae is defined as usual:

- for an atom $t \in \overline{\mathcal{HB}(\mathcal{F},\mathcal{P})}$, $H \models t$ iff $t \in H$,
- for a negative literal, $H \models \neg t$ iff $t \notin H$,
- for a rule body, $H \models l_1 \wedge \ldots \wedge l_n$ iff $H \models l_i$ for all $1 \leq i \leq n$,
- for a rule $h \leftarrow b$ containing variables $\mathcal{X} = \{X_1, \ldots, X_n\}$, $H \models h \leftarrow b$ iff $H \models h\sigma \leftarrow b\sigma$ for all substitutions $\sigma : \mathcal{X} \mapsto \overline{U(\mathcal{F})}$.[8] □

The semantics of a (negation-free) F-logic program P can now be defined as the minimal Herbrand interpretation H for which $H \models F$, called the *model* of P.[9]

In order to connect this model semantics of F-logic programs with the abstract Web model explained above (see Section 3), we need the notion of a *Web interface*.

Definition 4 (Web Interface) Let $\mathcal{URL}$ be the set of all URLs and R a set of reserved names (0-ary functors). A *Web Interface* $\mathcal{W}$ is a tupel $(R, explore)$, where $explore : \mathcal{URL} \rightarrow \mathfrak{P}(\overline{\mathcal{HB}(\mathcal{URL} \cup R, \emptyset)})$ is a function mapping URLs to sets of facts describing the contents of the document fetched in terms of the reserved names ($\mathfrak{P}(X)$ denotes the powerset of X). □

For the minimal schema informally introduced in Section 3, the set of reserved names R would be {url,get,webdoc,link,label,author,modif,type,error,hrefs,html}.

Definition 5 (Herbrand Web Interpretation) A *Herbrand Web interpretation* wrt. a Web interface $\mathcal{W}$ is a Herbrand interpretation $H \subset \overline{\mathcal{HB}(\mathcal{F} \cup R, \mathcal{P})}$ that additionally fulfills the *Web access axiom*:

$$\forall u \in \overline{U(\mathcal{F})}, \text{if } u : url, u.get \in H \text{ then } explore(u) \subset H.$$

that is, if the element $u.get$ representing the document belonging to the URL u is active in an interpretation, then all the information extracted by the analysis function *explore* is also part of the interpretation. A Herbrand Web interpretation H with $H \models P$ is called *Herbrand Web model* of P. □

Theorem 1 *Every negation-free program P has a unique minimal Herbrand Web model wrt. $\mathcal{W}$.* □

PROOF Web models of P are models of P. Thus, the intersection M of all Herbrand Web models is a model of P. As the *Web access axiom* holds in every Web model, it holds in the intersection as well. So M is a Web model of P. This establishes the uniqueness of the minimal Web model. The existence of Web models follows from the bottom-up construction below. Note, that minimal models may be infinite (cf. the rule (2.1)). ■

The *Web model* of a program P is the minimal Herbrand Web interpretation wrt. $\mathcal{W}$.

[8] Rules with complex molecules and multi-valued path expressions can be brought into this form, possibly by introducing new variables.

[9] The problem of choosing a model in the presence of negation or inheritance conflicts is dealt with in [KLW95].

5.2 Bottom-up Evaluation

If P is a pure Datalog program, the minimal model semantics has a quite natural procedural evaluation strategy (note that Datalog is a subset of F-logic). Beginning with the empty interpretation, the rules of the program are applied and the derived facts are added to the interpretation, until no new facts can be derived anymore. Formally, this means iterating the well-known operator T_P on sets of facts:

$$\begin{aligned} T_P(H) &:= \{h\sigma \mid h \leftarrow b \in P,\ \sigma : \mathcal{X} \to \overline{U(\mathcal{F})},\ H \models b\sigma\} \\ T_P^0 &:= \emptyset \\ T_P^{i+1} &:= T_P(T_P^i) \end{aligned}$$

Then, as T_P is monotone, the fixpoint T_P^∞ is the minimal model. For arbitrary F-logic programs P, $T_P(H)$ is in general not a Herbrand interpretation. To ensure this, we may extend the program by the closure axioms from Definition 2.[10] We denote the extended Program by P^*. So, the F-logic derivation operator is

$$T_P^F(H) := T_{P^*}(H).$$

Note, that due to the presence of functors and path constructors, the fixpoint may be infinite. To evaluate a Web model in a bottom-up way, we additionally have to ensure the *Web access axiom* from Definition 5. Therefore, we declare the Web derivation process by

$$T_P^{\mathcal{W}}(H) := T_P^F(H) \cup \bigcup\{explore(u) \mid u : url, u.get \in T_P^F(H)\} \cup H$$

Theorem 2 *The fixpoint $T_P^{\mathcal{W},\infty}$ is the minimal Herbrand Web model for a negation-free program P.* □

PROOF As P is negation-free and $T_P^{\mathcal{W},\infty} \supset T_P^{F,\infty}$, it is a model of P. We show the Web access axiom: If $u.get, u : url \in T_P^{\mathcal{W},\infty}$, there exists a minimal i such that $u.get, u : url \in T_P^{\mathcal{W},i}$. Due to the definition of $T_P^{\mathcal{W}}$, $explore(u) \subset T_P^{\mathcal{W},i+1} \subset T_P^{\mathcal{W},\infty}$. Minimality is shown as usual. ■

Implementation. As the FLORID prototype does not work on Herbrand interpretations but on semantic structures with a universe of oids, integrating Web access to the evaluation component simply amounts to a special treatment of the method get and the class url when adding new facts to the interpretation: Whenever the method get becomes defined for an object u, we have to check whether u is member of the class url; and whenever a new instance is added to the class url, we check whether the method get is defined for it. In both cases

[10] The set of axioms is infinite, but the number of axioms relevant for P is finite (there is an upper bound on the arity of predicates and method applications in P). For efficiency reasons, in the FLORID prototype, the closure properties are "hard-wired" into the Object Manager [FHKS97a].

the function *explore* is called for u and the resulting facts are added to the interpretation. As the web pages are only loaded when a *new* fact of this type is inserted, the web is incrementally accessed. In other words, no document is fetched more than once. Note that the Web interfacing part does not interfere with program optimization techniques.

As in the case of other Web languages, currently only text data of documents are utilized. However, our framework is capable to integrate (text) data drawn from other parts of documents as well, if the user enriches the *analyse* function with the necessary analyse tools.

6 Conclusion

In this paper, we have proposed a Web query language on top of F-logic, thus gaining a declarative semantics and a bottom-up evaluation algorithm. We have shown how relevant parts of the Web can be integrated into the local database, thereby allowing to restructure data and combine it with local information. The resulting evaluation algorithm has been integrated into the FLORID prototype.

We restricted our presentation to the basic requirements of HTML document analysis, but other document types (e.g. BibTeX) can be handled analogously. The real power of the *explore* cycle lies in the possibility to use more sophisticated analyse functions without changing the semantic framework. This a topic of future research.

As mentioned in the introduction, there is another thread of research activity besides Web query languages, namely the problem of handling *semi-structured data* which naturally appears in the Web context. One major part of this problem is to integrate heterogeneous data from different sources. Due to its meta features, F-logic is well suited to such tasks and has been already used for it, see e.g. [CRD94]. In future research, we will investigate the handling of semi-structured data in F-logic, which would allow to join the above-mentioned two streams of research into a common framework.

References

[Abi97] S. Abiteboul. Querying Semi-Structured Data. In *Proc. Intl. Conference on Database Theory (ICDT)*, number 1186 in LNCS, pp. 1–18. Springer, 1997.

[AQM+] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *The Journal of Digital Libraries.* to appear. ftp://db.stanford.edu//pub/papers/lorel96.ps.

[AV97] S. Abiteboul and V. Vianu. Queries and Computation on the Web. In *Proc. Intl. Conference on Database Theory (ICDT)*, number 1186 in LNCS, pp. 262–275. Springer, 1997.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrandt, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. ACM Intl. Conference on Management of Data (SIGMOD)*, 1996.

[Cat94] R. Cattell. *The Object Database Standard: ODMG-93.* Morgan Kaufmann, 1994.

[CRD94] Y. Chang, L. Raschid, and B. Dorr. Transforming queries from a relational schema to an equivalent object schema: a prototype based on F-logic. In *International Symposium on Methodologies in Information Systems, (ISMIS-94)*, October 1994.

[FHK+97] J. Frohn, R. Himmeröder, P.-T. Kandzia, G. Lausen, and C. Schlepphorst. FLORID: A Prototype for F–Logic. In *Proc. Intl. Conference on Data Engineering (ICDE)*, 1997.

[FHKS97a] J. Frohn, R. Himmeröder, P.-T. Kandzia, and C. Schlepphorst. FLORID, Version 2.0, User Manual, 1997. Available from ftp://ftp.informatik.uni-freiburg.de/pub/florid/manual.ps.gz.

[FHKS97b] J. Frohn, R. Himmeröder, P.-T. Kandzia, and C. Schlepphorst. How to Write F-Logic Programs in FLORID, Version 2.0, 1997. Available from ftp://ftp.informatik.uni-freiburg.de/pub/florid/tutorial.ps.gz.

[FLU94] J. Frohn, G. Lausen, and H. Uphoff. Access to Objects by Path Expressions and Rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. Intl. Conference on Very Large Data Bases (VLDB)*, Santiago de Chile, 1994.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, July 1995.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A Query System for the World-Wide Web. In *Proc. Intl. Conference on Very Large Data Bases (VLDB)*, 1995.

[LSS96] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A Declarative Language for Querying and Restructuring the Web. In *Proc. Sixth International Workshop on Research Issues in Data Engineering (RIDE'96)*, 1996.

[MM97] A. Mendelzon and T. Milo. Formal Models of Web Queries. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, 1997.

[MMM96] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. In *Proc. of 5th International Conference on Parallel and Distributed Information Systems (PDIS'96)*, 1996.

[Suc97] D. Suciu, editor. *Proc. of the Workshop on Management of Semi-Structured Data (in conjunction with SIGMOD/PODS)*, Tucson, Arizona, 1997. http://www.research.att.com/~suciu/workshop-papers.html.