Querying the Web with FLORID

Rainer Himmeröder Bertram Ludäscher

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

1 Introduction

Models and languages for querying the Web, for handling semistructured data, and for integration and restructuring of information have recently attracted a lot of interest [MV98]. We argue that *dood* languages, i.e., supporting deductive and object-oriented features, are particularly suited in this context: *Object-orientation* provides a flexible common data model for combining information from heterogeneous sources and for handling partial information. Techniques for navigating in object-oriented databases can be applied to semistructured databases as well, since the latter may be viewed as (very simple) instances of the former. *Deductive rules* provide a powerful framework for expressing complex queries in a high-level, declarative programming style. WebLog [LSS96] and ADOOD [GMNP97], for example, build upon the dood language F-logic [KLW95]. FLORID [FLO] is an implementation of F-logic, which has been extended to provide a declarative semantics for querying the Web [HLLS97, HLLS98]. The proposed extension allows extraction and restructuring of data from the Web and a seamless integration with local data. A main advantage of the approach is that it brings together the above-mentioned issues in a unified, formal framework and supports rapid prototyping and experimenting with all these features. In particular, FLORID programs may be used (simultaneously) as wrappers, mediators, or for "ordinary" queries. In this paper, we focus on querying the Web; for the management of semistructured data with FLORID, see [HLM+].

2 Preliminaries: F-Logic in a Nutshell

We briefly review the basic constructs of F-logic and its extension by path expressions. Consider the following fragment of an F-logic program:

```
person[name ⇒string; children@(integer) ⇒⇒person].      % signature of class person
employee::person.                                        % subclass relationship

john:employee[                                           % instance relationship and
      name →"John Smith"; children@(1998) →→{mary,bob}]. % ... some example data

X.father:man  ←X:person.                                 % object creation by ...
X.mother:woman ←X:person.                                % ... path expressions
```

First, the *signature* of class person is specified: The *single-valued* method name yields instances of class string, whereas the *multi-valued* (and parameterized) method children yields instances of person. The *subclass* relation employee::person states that all members of employee are also members of person. Next, john:employee defines that the object named john is an *instance* of class employee; the specification inside [...] defines the actual data values for name and children.

The last two rules demonstrate how path expressions in the head can be used to create new object identifiers (oids): If X is bound to an instance of person, then the single-valued method father becomes defined for X. The newly "created" father is referenced by the *path expression* X.father and is made an instance of man. In particular, john[father→john.father] and

(john.father):man hold (similarly for mother and woman). Thus, the dot "." corresponds to navigation along single-valued methods (→) like name and father, while ".." is used to navigate along multi-valued methods (↠) like children, e.g., as in john..children@(Y)[surname→john.surname].

The use of path expressions for object creation is crucial for our approach to data-driven exploration of the Web using F-logic. Object creation has to be used with care in order to avoid infinite universes and nontermination: e.g., if a rule X.father:man ← X:man were added to the program, an infinite number of objects like john.father, john.father.father, etc. would be created.

3 Exploring the Web with F-Logic

The Web Model. As usual, we conceive the Web as a graph-like structure consisting of documents and links between them. More precisely, we distinguish between the class url of (potential) urls, and webdoc of (accessed) Web documents (Fig. 1). Urls are instances of string, for which a special method get is definable (see below). Typical elements of class webdoc are HTML pages, but other document types may also be included (e.g., BibTEX). In particular, one may define a subclass sgmldoc such that the parse-tree of fetched SGML documents can be analyzed:[1] If a Web document has been accessed, a number of *system-defined* methods may become defined for it: e.g., url (the url of the Web document), author, modif (time of last modification), type, and, most notably, the multi-valued method hrefs@(*label*) representing the outgoing links of the Web document (Fig. 1). Note that hrefs is parameterized with the label of the link. If the Web access fails, error returns the reason of the failure (e.g., *page not found*).

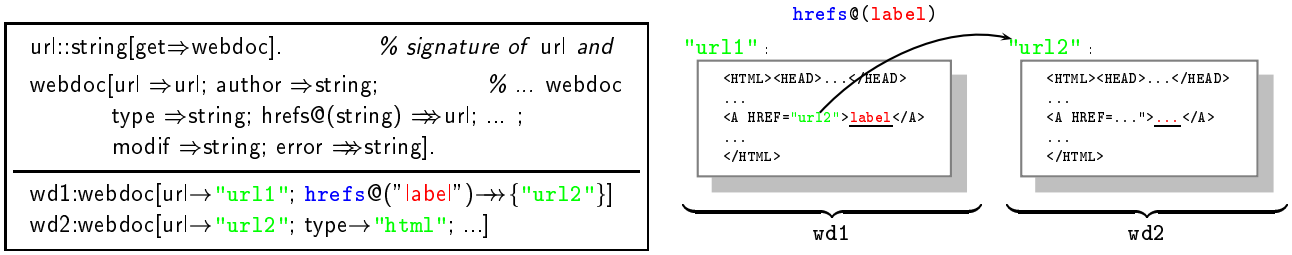

Figure 1: F-logic Web model: signature and example data

Data-Driven Exploration. A Web document is accessed and added to the local F-logic database by defining the method get for an instance u of class url in the head of a rule, thereby creating the new oid "u.get" of the fetched Web document. After the oid u.get has been created, system-defined methods are automatically "filled in" by FLORID, and the Web document named u.get becomes an ordinary F-logic object (conceivable as a large string). Thus, u.get is "cached" and the url u is accessed only once. Note that the *potentially* system-defined methods for class webdoc are given by the FLORID system—the *actually* system-defined (i.e., "filled") methods for u.get depend on the result of accessing the url u. For example, if error is defined, then hrefs@(*label*) will be undefined. Here, the advantages of using an object-oriented framework like F-logic become apparent: although the instances of a certain class may typically define a certain set of methods, some (or all) of these methods may be omitted. Thus, the use of NULL values as in the relational model can be avoided.

Apart from this first document analysis done by the system, the main power of the approach lies in the possibility to specify arbitrary *user-defined* methods for fetched documents using all features of F-logic (and path expressions). Consider, e.g., the following F-logic program:

[1] This feature is not yet implemented in FLORID.

```
("http://www.informatik.uni-freiburg.de/~dbis/":url).get.    % (1) define and access start url
U:explored ← (U:url).get.                                    % (2) remember explored documents
U1.get ← (U:explored).get[hrefs@(Lbl) →→{U1}],               % (3) transitively access url U1 ...
         substr(U,U1).                                       % ... if U is a prefix of U1
```

The path expression in (1) defines a particular string as an instance of url and, since the method get is defined for this url, accesses the corresponding document. Due to (2), the urls of all accessed documents become instances of class explored. (3) uses the power of deduction with recursion and transitively accesses all documents reachable from an already explored url U, provided that the referenced url U1 is a substring of U. Thus, only urls starting with http://...~dbis will be accessed; for other links, get remains undefined. Therefore, explored and unexplored urls can be distinguished using the queries ?-U:explored and ?-U:url, not U:explored, respectively. In particular, only a finite number of new oids u.get is created.

Since exploration of the Web is completely data-driven, a seamless integration with the bottom-up evaluation strategy of FLORID and a declarative semantics of the language is achieved; see [HLLS97] for the details.

4 Querying Web Data with FLORID: CIA World Factbook

We illustrate the features of our approach for querying and restructuring Web data, using the pages of the *CIA World Factbook*, a collection of Web pages providing information about the countries in the World (e.g., on geography, people, government, and economy). Although one may argue that the data in the World Factbook is highly regular and should be put on the Web as a database in the first place, there are some idiosyncrasies and irregularities to consider (see below for a simple example, i.e., the "pseudo-values" of the attribute capital). Moreover, the actual database is *not* available from the Web, whereas the semistructured HTML pages are.

Extracting Data. Note that the following rules make up a *self-contained* program, i.e., there are no separate languages for wrappers or mediators:

First, the urls of the World Factbook homepage, of the page for countries in Europe, and of a local mirror are defined for the object cia, our "root" object for the Factbook:

```
cia[world →"http://www.odci.gov/cia/publications/nsolo/factbook/global.htm" ;
    europe →"http://www.odci.gov/cia/publications/nsolo/factbook/eur.htm" ;
    mirror →"file:/home/dbis/flogic/data/ciawfb/global.htm" ].
```

To allow for easy substitution of the data source, a generic name cia.src is defined:

```
cia[src →cia.world].    % use the main Factbook page here
```

Alternatively, cia.src may be set to cia.europe, or may even be rule-defined: e.g., if access to cia.world fails, cia.src can be defined as cia.mirror.

The string represented by cia.src is made an instance of class url and accessed via get:

```
(cia.src:url).get.      % ⇔ (cia.src):url ∧ (cia.src).get
```

Thus, cia.src.get is the name (logical oid) of the accessed Web document and hrefs@(*label*) is defined for it by the system (unless an error has occurred). The source page cia.src.get of the Factbook contains links to the individual country pages. These links are used to populate the class country with instances C and the urls U of C:

```
C:country[url →U] ←                  % remember the url U of country C after ...
    cia.src.get[hrefs@(Lbl) →→{U}],  % ... extracting all labels Lbl and urls U ...
    match(Lbl, "\(.*\) ([0-9]","\1", C).   % ... and removing excess parts from Lbl
```

The labels Lbl in cia.src.get are the names of the countries followed by the size of the page in KBytes (e.g., "Spain (32 KB)"). Here, the built-in predicate match is used to strip off this size information: e.g., match("Spain (32 KB)", ..., ..., C) yields C="Spain".

match(*Str*, *RegEx*, *Fmt*, *Res*) finds all strings in the input string (or Web document) *Str* which match the pattern given by the regular expression *RegEx*. The format string *Fmt* describes how the matched strings should be returned in *Res*. This feature is particularly useful when using groups (expressions enclosed in \(...\)) in regular expressions.

The individual pages of the extracted countries are accessed by defining get for the corresponding urls as usual:

```
U.get ← C:country[url→U].   % retrieve all country pages
```

Observe that the name url can be used for both, the predefined class of urls, and the user-defined method url: In F-logic, also methods and classes are objects; thus, it is possible to reason about schema information.

Finally, data from the country pages can be extracted and stored in the F-logic database. For example, among lots of other data, the following can be extracted (note that the match predicate treats Web documents as strings):

```
C[capital →X]          ←  match(C:country.url.get, "Capital:.*\n\(.*\)", "\1", X).
C[total_area →X]       ←  match(C:country.url.get, "total area:.*\n\(.*sq km\)", "\1", X).
                          ⋮
C[external_debt →X]    ←  match(C:country.url.get, "External debt:.*\n\(.*\)", "\1", X).
```

Querying the Data. Once the data has been extracted, it can be queried, restructured, and integrated with data from other sources, using all features of F-logic and FLORID, respectively:

```
?- N = count{C ; C:country}.  % (Q1) use aggregation to count the countries
?- C:country[capital→CA].     % (Q2) name all countries and their capitals!
?- C:country, not C.capital.  % (Q3) which countries do not have a capital?
```

For cia.src=cia.world, query (Q1) yields N=266 countries. However, (Q2) outputs a binary relation (Country,Capital) with only 256 entries. (Q3) reveals the 10 "countries" for which the method capital is not defined, e.g., "Antarctica", "Atlantic Ocean", and "World". It turns out that there are some more "countries" which have the method capital defined, yet do *not* have a proper capital. For example, the fact

```
"Bouvet Island":country[capital→"none; administered from Oslo, Norway"]
```

can be derived by FLORID. Thus, we may specify the class of *real* countries as follows:

```
C:real_country ← C:country[capital→CA], not substr("none", CA).
```

Now, the query ?- C:country, not C:real_country discloses 25 more "false countries" (apart from the 10 above) including, e.g., "Bouvet Island", "Clipperton Island", and "Western Sahara".

Schema Browsing. Since method and class names are first-class citizens in F-logic, reasoning about schema information is possible. Consider the following queries:

```
?- _:country[M→_].            % (Q4) what methods are defined for countries?
?- _:country.M, C:country, not C.M.  % (Q5) return countries with undefined methods
```

Query (Q4) yields all single-valued methods (capital, total_area, land_area, etc.) potentially defined for countries (i.e., defined for at least *some* country). The different occurrences of the anonymous (don't care) variable "_" denote *distinct* $\exists$-*quantified* variables. The first literal _:country.M of (Q5) is a syntactic variant of (Q4); together with the rest of (Q5), "countries" C with undefined methods M are reported: e.g., for C="Ashmore and Cartier Islands", the method M="labor_force" is undefined.

Integration and Restructuring. Data from heterogenous sources, for instance accessed via http/ftp/file protocols, or from the local F-logic database, are easily integrated: E.g., by adding F-logic facts of the form

"Spain"[continent→"Europe"]. "Tunisia"[continent→"Africa"]. ...

additional information regarding the continents of countries can be defined using the local database. Alternatively, we can use the continents from the cia.world page and all countries reachable from the corresponding continent page: First, we define the continents and their urls:

"Europe":continent[url→"http://www.odci.gov/cia/publications/nsolo/factbook/eur.htm"].
"Asia":continent[url→"http://www.odci.gov/cia/publications/nsolo/factbook/asia.htm"].
...

Next, assuming that we have already fetched the continent pages via get, we use the following rule to extract from the continent pages all countries of the corresponding continent:

```
CT:continent[countries→↠{C}] ←
    CT:continent.url.get[hrefs@(Lbl) →_], match(Lbl, "\(.*\) ([0-9]","\1", C).
```

Then, instead of adding local facts about continent explicitly (as above), the method continent for instances of class countries can be defined by rules:

C:country[continent→CT] ← CT:continent[countries→↠{C}].

In this way, data from different sources can be "attuned" and checked for inconsistencies. Moreover, restructuring information into different classes (like country, real_country, continent, etc.) is easy to accomplish due to F-logic's flexible rule language which supports variables at method and class positions.

References

[BRR97] F. Bry, K. Ramamohanarao, and R. Ramakrishnan, editors. *Intl. Conference on Deductive and Object-Oriented Databases (DOOD)*, number 1341 in LNCS, Montreux, Switzerland, 1997. Springer.

[FLO] The FLORID Home Page. http://www.informatik.uni-freiburg.de/~dbis/florid/.

[GMNP97] F. Giannotti, G. Manco, M. Nanni, and D. Pedreschi. Datalog++: A Basis for Active Object-Oriented Databases. In Bry et al. [BRR97].

[HLLS97] R. Himmeröder, G. Lausen, B. Ludäscher, and C. Schlepphorst. On a Declarative Semantics for Web Queries. In Bry et al. [BRR97], pp. 386–398.

[HLLS98] R. Himmeröder, G. Lausen, B. Ludäscher, and C. Schlepphorst. *FLORID*: A DOOD-System for Querying the Web. In *Demonstration Session at EDBT*, Valencia, Spain, 1998.

[HLM+] R. Himmeröder, B. Ludäscher, W. May, C. Schlepphorst, and G. Lausen. Managing Semistructured Data with *FLORID*: A Deductive Object-Oriented Perspective. Manuscript.

[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.

[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.

[MV98] U. Masermann and G. Vossen. Suchmaschinen und Anfragen im World Wide Web. *Informatik Spektrum*, 21(1):9–15, 1998.