Processing Xml–Documents in Prolog

Dietmar Seipel

University of Würzburg, Dept. of Computer Science
Am Hubland, D – 97074 Würzburg, Germany
seipel@informatik.uni-wuerzburg.de

Abstract. In this paper we will show how *semi–structured data* such as Xml–data can be represented and queried nicely in a logic programming environment. We will introduce a document object model for Xml–documents in Prolog, which we call *field–notation*. It simplifies dealing with semi–structured data; instead of accessing a component of a Prolog–fact by its argument position it becomes possible to access the component by its attribute name, or to access nested components by a complex *path* or *tree expression*.
We have implemented a library FnPath of predicates dealing with Xml–data in field–notation, and we have done several practical case studies for extracting and transforming information from Xml–documents in a declarative style. The relevant information is located automatically in the document, such that the extraction process becomes robust to certain changes of the document structure over time.

Keywords.
Xml, semi–structured data, Prolog, query languages, deductive databases

1 Introduction

Query languages for object–oriented databases have been investigated since about 15 years, cf., e.g., the query language Oql of O_2 [2], the language Lorel [1], and the logic–based language *F–Logic* [8] and its successor languages. Since the emergence of Xml the goal was to adapt these query languages for processing Xml–data, cf. Xql (1998), Xml–Ql (1998), Xml Query (also known as Quilt or XQuery), the extended style sheet language Xsl (1999), and the pattern–based language Xcerpt [4]. Many of the Xml–query languages including the emerging W3C standard Xml Query [12] are based on the path language XPath (1999). Recently also specialized Xml–databases, such as, e.g., Tamino, have been developed. Xml is a well–known, self–describing data format, and it is often considered as the future of the World Wide Web. Xml–data can be considered as *semi–structured* data, since their schema or DTD might be unknown or frequently changing [11].

The goal of the *Semantic Web* is to add a *logic component* for *reasoning* about the Web, cf. [3]. The *RuleML–initiative* is working on a standard for representing rule systems on the Web. The logic programming language Prolog can handle symbolic computations on complex, tree–structured objects nicely, and it can be used for reasoning about semi–structured data.

The following *characteristics of* PROLOG are important when we consider representing relational or semi–structured data in PROLOG. The usual representation would be that the relation name becomes a predicate with the components of a tuple as its arguments. E.g., a relation stock with attributes Index, Wkn, Date, and Value can be represented by PROLOG–facts of the form `stock(Index,Wkn,Date,Value)`. The schema information (i.e., the attribute names) is not stored in the PROLOG–facts. There are the following implications of this representation: Firstly, the *selection* of attribute values is by argument position, which is problematic if there are many arguments. Secondly, the update of components within a structure `Stock`, e.g. the assignment `Stock.Value := 60`, is not possible without creating a new structure `Stock'`, because the *assignment* of variables is non–destructive in PROLOG.

The rest of this paper is organized as follows: In Section 2 we will define a *field–notation* for representing XML–data as terms in a logic programming environment, and we will show how these data can be accessed nicely avoiding the above mentioned problems. In Section 3 an extensible library FNPath for processing data in field–notation is introduced, and the semantics of FNPath is described briefly. Some practical applications of FNPath are reported in Section 4; in particular, it is shown how XML–data in field–notation can be extracted from HTML–files.

2 Complex Objects in PROLOG

Syntactically, both XML and PROLOG are based on nested *term structures*. We will show how XML–structures can be represented and queried in PROLOG. Since XML can be used for representing complex objects, cf. [1], this gives us a way of handling complex objects in PROLOG or deductive databases.

2.1 XML–Objects in Field–Notation

Complex Objects in PROLOG. We will use *association lists* for representing complex objects, cf. [1], in PROLOG. This data structure is familiar to LISP–programmers. The following inductive definition of association lists uses the infix operator ":" for pairing an attribute with its associated value.

Definition 1 (Association Lists).

1. If $\mathsf{a_i}$ and $\mathsf{v_i}$ $(1 \leq i \leq n)$ are PROLOG–terms, then $[\mathsf{a_1} : \mathsf{v_1}, \ldots, \mathsf{a_n} : \mathsf{v_n}]$ is an association list.
2. Each $\mathsf{a_i}$ is called an attribute, and $\mathsf{v_i}$ is called the associated value.

Observe that association lists are PROLOG–terms themselves; thus, it is possible to have nested association lists, where some of the values $\mathsf{v_i}$ are association lists.

For example, $\mathsf{As} = [\mathsf{f}(1,\mathsf{a}) : 7, \mathsf{b} : [\mathsf{c} : 1]]$ is an association list with the attributes $\mathsf{f}(1,\mathsf{a})$ and b, and with the corresponding values 7 and $[\mathsf{c} : 1]$, respectively. $[\mathsf{c} : 1]$ is itself an association list, wheras 7 is just a PROLOG–constant.

A complex object O with sub–objects O_i, $1 \le i \le n$, that are selected by attributes a_i (i.e., $\mathsf{O}_\mathsf{i} = \mathsf{O}.\mathsf{a}_\mathsf{i}$) can be represented as an association list $\mathsf{O}' = [\mathsf{a}_1 : \mathsf{O}'_1, \ldots, \mathsf{a}_\mathsf{n} : \mathsf{O}'_\mathsf{n}]$, where O'_i is the association list for O_i. A complex object without any sub–objects is represented as a constant in PROLOG.

Association lists have got several advantages when compared to ordinary PROLOG–facts of the form $\mathsf{o}(\mathsf{O}'_1, \ldots, \mathsf{O}'_\mathsf{n})$: Firstly, the sequence of attribute/value–pairs $\mathsf{a}_\mathsf{i} : \mathsf{O}'_\mathsf{i}$ is arbitrary. Secondly, values O'_i can be accessed by attributes a_i rather than by argument positions. Thirdly, the database schema can be changed at run time. Fourth, null values can be omitted, and new values can be added at run time. Fifth, XML–objects in field–notation can be queried and modified very elegantly.

XML–*Objects in* PROLOG. The following XML–example shows a database with information about the charts of the German stock index dax100; each chart has an identifying attribute wkn and entries with the attributes date and value:

```
<stocks index="dax100">
   <chart wkn="200400">
      <entry date="14.12.2002" value="30"/>
   </chart>
   <chart wkn="600800">
      <entry date="14.12.2002" value="40"/>
      <entry date="15.12.2002" value="50"/>
   </chart>
</stocks>
```

An XML–object $\mathsf{X} = \langle \mathsf{T}\ \mathsf{a}_1 = \text{"}\mathsf{v}_1\text{"} \ldots \mathsf{a}_\mathsf{n} = \text{"}\mathsf{v}_\mathsf{n}\text{"} \rangle \ldots \langle /\mathsf{T} \rangle$ with the tag "T" can be represented as a PROLOG–term $\mathsf{T}:\mathsf{As}:\mathsf{C}$, where $\mathsf{As} = [\mathsf{a}_1 : \text{'}\mathsf{v}_1\text{'}, \ldots, \mathsf{a}_\mathsf{n} : \text{'}\mathsf{v}_\mathsf{n}\text{'}]$ is an association list for the attribute/value–pairs and C represents the contents (i.e., the sub–elements) of X.[1] We call $\mathsf{T}:\mathsf{As}:\mathsf{C}$ an FN–object. For example, the XML–document above can be represented by the following FN–object:

```
stocks:[index:dax100]:[
   chart:[wkn:200400]:[
      entry:[date:'14.12.2002', value:30]:[] ],
   chart:[wkn:600800]:[
      entry:[date:'14.12.2002', value:40]:[],
      entry:[date:'15.12.2002', value:50]:[] ] ]
```

[1] We arbitrarily assume that ":" is right associative, i.e., $\mathsf{T} : \mathsf{As} : \mathsf{C}$ means $\mathsf{T} : (\mathsf{As} : \mathsf{C})$.

Definition 2 (Field–Notation).

1. If T and C are Prolog–terms and As is an association list, then $\mathsf{O} = \mathsf{T} : \mathsf{As} : \mathsf{C}$ is called an FN–object with the tag T and the contents C.
   – If $\mathsf{As} = [\mathsf{a}_1 : \mathsf{v}_1, \dots, \mathsf{a}_n : \mathsf{v}_n]$, then each a_i is called an attribute of O and v_i is called the corresponding value.
   – If $\mathsf{As} = [\,]$, then O can alternatively be represented as $\mathsf{T} : \mathsf{C}$.
2. Given an FN–object $\mathsf{O} = \mathsf{T} : \mathsf{As} : \mathsf{C}$, such that $\mathsf{C} = [\mathsf{c}_1, \dots, \mathsf{c}_m]$ is a list of FN–objects, then each c_i is called a sub–element of O.

Observe that FN–objects are Prolog–terms themselves.

2.2 Access to Components in Field–Notation

Selection. Given an FN–object O and two Prolog–terms A and X. The binary infix–predicate ":=" allows for accessing the sub–elements and the attribute values of O, which corresponds to the *child–axis* and the *attribute–axis*, respectively, of XPath.

– The call $\mathsf{X} := \mathsf{O}\hat{}\mathsf{A}$ succeeds for all most general substitutions θ, such that $\mathsf{X}\theta$ is a sub–element with the tag $\mathsf{A}\theta$ of $\mathsf{O}\theta$.
– The call $\mathsf{X} := \mathsf{O@A}$ succeeds for all most general substitutions θ, such that $\mathsf{X}\theta$ is the value of the attribute $\mathsf{A}\theta$ of $\mathsf{O}\theta$.

This reminds of the evaluation of arithmetic expressions in Prolog, X is $3*(4+5)$, where "is" is an infix–predicate symbol with two arguments, which evaluates the arithmetic term $3*(4+5)$ and assigns the result to the first argument "X".

The predicate ":=" can handle *complex path expressions* for selecting sub–components. E.g., we get:

```
?- O = stocks:[index:dax100]:[
          chart:[wkn:200400]:[
             entry:[date:'14.12.2002', value:30]:[]]],
   X := O^chart, Y := O@index.
   Z := O^chart@wkn.

X = chart:[wkn:200400]:[
       entry:[date:'14.12.2002', value:30]:[]],
Y = dax100,
Z = 200400
```

In pure Prolog the selection with the complex path expression `^chart@wkn` would look much more complicated:

```
?- O = stocks:[index:dax100]:[
          chart:[wkn:200400]:[entry:[...]:[]]],
   O = _:_:X, member(chart:Y:_,X), member(wkn:Z,Y).

Z = 200400
```

Assignment. We can assign new values to attributes or elements as follows:

```
?- O = stocks:[index:dax100]:[
          chart:[wkn:200400]:[
             entry:[date:'14.12.2002', value:30]:[]]],
   X := O^chart*[^entry@value:40].

X = chart:[wkn:200400]:[
      entry:[date:'14.12.2002', value:40]:[]].
```

X is obtained by first selecting the sub–element with the tag chart, and by then modifying the value–attribute of the embedded entry–element. Observe, that we have to create a new object X holding the result, since in Prolog there exists no destructive assignment.

3 The Library FnPath

In this section we will describe the selection and the transformation of sub–elements of FN–objects using complex tree expressions. They are the *basic methods* that we have implemented within a Prolog–library FnPath for dealing with FN–objects.

We have extended the library FnPath by additional, more *advanced methods*, such as the selection/deletion of all elements/attributes of a certain pattern, the transformation of sub–elements according to substitution rules, and the manipulation of path or tree expressions. Their usage can be seen in Section 4.

3.1 Tree Expressions and Query Terms

Path expressions and tree expressions are defined inductively as strings over Prolog–terms and some other characters.

Definition 3 (Paths, Selection Trees).

1. If t is a Prolog–term, then $\hat{}t$ and $@t$ are paths. If π_i $(1 \leq i \leq n)$ are paths, then $\pi_1 \ldots \pi_n$ is a path.
2. Every path is a selection tree. If π is a path and τ_i $(1 \leq i \leq n)$ are selection trees, then $\pi - [\tau_1, \ldots, \tau_n]$ is a selection tree.

For example, `^chart-[@wkn,^entry-[@date,@value]]` is a selection tree. If O is an FN–object and τ is a selection tree, then O τ can be considered as a Prolog–term. Depending on their position, the functors ^ and @ are either perceived as right–associative, binary infix functors or as unary prefix functors with suitable precedences and types (cf. Section 3.2).

A path is applied to an FN–object O by iteratively applying the selectors, and a selection tree $\pi - [\tau_1, \ldots, \tau_n]$ is applied by first applying π to O and by then applying the selection trees τ_i in parallel. Thus, a query with *multiple selection paths* returns a list of objects, namely one object for each selector:

```
?- O = stocks:[index:dax100]:[
          chart:[wkn:200400]:[
             entry:[date:'14.12.2002', value:30]:[]]],
   X := O^chart-[@wkn, ^entry-[@date, @value]].

X = [200400, ['14.12.2002', 30]].
```

If a path contains *variable symbols*, then all instances of the path which are an allowed path in the query object can be generated on backtracking:

```
?- O = stocks:[index:dax100]:[
          chart:[wkn:200400]:[
             entry:[date:'14.12.2002', value:30]:[] ] ],
   X := O^chart^entry@Path.

X = '14.12.2002', Path = date ;
X = 30, Path = value

?- O = stocks:[index:dax100]:[
          chart:[wkn:200400]:[
             entry:[date:'14.12.2002', value:30]:[] ] ],
   30 := O^Path.

Path = chart^entry@value
```

If a selection tree τ is not *compatible* with an FN–object O, then O τ does not return any results. E.g., $\tau =$`^entry` is not compatible with our complex object, since O does not contain any (direct) sub–elements with the tag `chart`. Obviously, a path π containing two subsequent functors `@`, such as, e.g., `@a@b`, is not compatible with XML–objects O in field–notation, since XML–attributes cannot have a complex structure.

Definition 4 (Association Trees, Assignment Trees).

1. If π_i is a path and v_i is a PROLOG–term ($1 \leq i \leq n$), then

$$\pi_0 - [\pi_1 : v_1, \ldots, \pi_n : v_n]$$

   is an association tree. If π_0 is a path and τ_i ($1 \leq i \leq n$) are association trees, then $\pi_0 - [\tau_1, \ldots, \tau_n]$ is an association tree.
2. If π_0 is a path and τ_i ($1 \leq i \leq n$) are association trees, then

$$\pi_0 * [\tau_1, \ldots, \tau_n]$$

   is an assignment tree.

An assignment tree $\pi_0 * [\tau_1, \ldots, \tau_n]$ selects a sub–element of an FN–object O, and then it iteratively applies the association trees τ_i to change certain attributes or elements. It is possible to change components at arbitrary depth in the document.

```
?- O = stocks:[index:dax100]:[
          chart:[wkn:200400]:[
             entry:[date:'14.12.2002', value:30]:[]]],
   X := O^chart*[^entry@value:40].

X = chart:[wkn:200400]:[
       entry:[date:'14.12.2002', value:40]:[]].
```

Selection trees and assignment trees can be combined within *FN–trees*, which form the basis for FN–query atoms.

Definition 5 (FN–Trees, FN–Query Atoms).

1. Both selection trees and assignment trees are FN–trees. If π_0 is a path and τ_i $(1 \leq i \leq n)$ are FN–trees, then $\pi_0 - [\tau_1, \ldots, \tau_n]$ is an FN–tree.
2. If O is an FN–object, τ is an FN–tree, and X is a Prolog–term, then $\mathsf{O}\ \tau$ is an FN–query term and $\mathsf{X} := \mathsf{O}\ \tau$ is an FN–query atom.

An FN–tree represents a *labelled tree*, where each edge is labelled by a path. The nodes can be labelled by $*$ or by a Prolog–term. Each path from the root to a leaf contains at most one node that is labelled by $*$; in this case the leaf is labelled by a Prolog–term. The FN–tree that is used in the following example is given in Figure 1:

```
?- O = stocks:[index:dax100]:[
          chart:[wkn:200400]:[
             entry:[date:'14.12.2002', value:30]:[]]],
   Y := O-[@index, ^chart^entry*[@value:40]].

Y = [dax100, entry:[date:'14.12.2002',value:40]:[]].
```

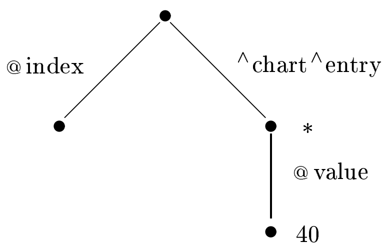

Fig. 1. An FN–Tree

3.2 Evaluation of FN–Query Atoms in Prolog

We have implemented a Prolog–engine for evaluating FN–query atoms, cf. [10]. The operators ^ and @ are defined as right–associative, binary infix operators with the same precedence, and at the same time as unary prefix operators using the call op(Precedence,Type,Name) for the following parameters:

Name	Precedence	Type
^	200	xfy
^	100	fx
@	200	xfy
@	100	fx

Simplified Notations. A selection tree or an association tree $\pi - [\tau_1, \ldots, \tau_n]$ where all τ_i start with the same selector $\circ$, i.e., $\tau_i = \circ\tau'_i$, can equivalently be represented as $\pi \circ [\tau'_1, \ldots, \tau'_n]$. An assignment tree $\pi{*}[\tau_1, \ldots, \tau_n]$ where all τ_i start with $^\wedge$, i.e., $\tau_i = {^\wedge}\tau'_i$, is represented as $\pi * [\tau'_1, \ldots, \tau'_n]$. For example, `O-[^a,^c]` can be represented as `O^[a,c]`, and `O*[^a:5,^c:6]` can be represented as `O*[a:5,c:6]`.

Aggregation Operators and Embedded Computations. We can handle query terms $\mathsf{X} := \mathsf{O} - \mathsf{agg}(\mathsf{X}_1, \ldots, \mathsf{X}_n, \pi)$ with *aggregation goals* with parameters $\mathsf{X}_1, \ldots, \mathsf{X}_n$, such as `O-average(^a)`, `O-sum(^a)`, and `O-nth(2, ^a)`. Here, the path π is first applied to the object O by calling $\mathsf{Y} := \mathsf{O}\ \pi$, and then the aggregate X is obtained from the list Ys of all results Y of this call by calling $\mathsf{agg}(\mathsf{X}_1, \ldots, \mathsf{X}_n, \mathsf{Ys}, \mathsf{X})$. Arbitrary user–defined PROLOG–predicates can be used for agg. For example, the query `X := O-agg(^a)` returns the list of all a–sub–elements of O, if agg is defined by the fact agg(Ys, Ys). The following query returns the average of all value–attributes within the second chart–element:

```
?- O = stocks:[index:dax100]:[
          chart:[wkn:200400]:[
             entry:[date:'14.12.2002', value:30]:[]],
          chart:[wkn:600800]:[
             entry:[date:'14.12.2002', value:40]:[],
             entry:[date:'15.12.2002', value:50]:[]]],
   X := O-nth(2, ^chart)-average(^entry@value).

X = 45
```

Finally, arbitrary *embedded computations* – such as computing the changes from one day to the next day by a user–defined function changes – can be started while evaluating a query term:

```
?- O = ..., X := O--changes.

X = stocks:[index:dax100]:[
       chart:[wkn:200400]:[ ],
       chart:[wkn:600800]:[
          entry:[from:'14.12.2002', to:'15.12.2002',
             change:'+25%']:[]]]
```

3.3 Related Concepts

We will briefly relate FnPath to two representative concepts for dealing with Xml–data: Xml Query and F–Logic. FnPath differs from Xml Query and from F–Logic in the built–predicates, which are allowed in FN–trees, and in the library of built–in predicates, which we have implemented.

Xml *Query.* The following Prolog–rule uses the features of FnPath for computing all stock values for a given day `Date`:

```
stock(Stocks,Date,entries:[]:Entries) :-
   findall( entry:[wkn:Wkn, value:Value]:[],
      ( Chart := Stocks^chart,
        Wkn   := Chart@wkn,
        Entry := Chart^entry,
        Date  := Entry@date,
        Value := Entry@value ),
      Entries ).
```

It can be applied to the FN–object `Stocks` given in Section 2.1. The corresponding FLWR–expression in Xml Query would look like follows:

```
<entries>
   { for $Chart in document("http://www.stocks.de")/stocks/chart
     let $Wkn = $Chart@wkn
     for $Entry in $Chart/entry
     let $Value = $Entry@value
     where Date = $Entry@date
     return <entry wkn="$Wkn" value="$Value"/> }
</entries>
```

F–Logic. F–Logic is also based on complex, nested structures, such as

```
stocks[ index -> dax100,
   charts --> {
      [ wkn -> 200400,
        entries --> {
           [ date -> '14.12.2002', value -> 30 ] } ],
      [ wkn -> 600800,
        entries --> {
           [ date -> '14.12.2002', value -> 40 ],
           [ date -> '15.12.2002', value -> 50 ] } ] } ]
```

which are called molecules; but a complex object is usually represented by a collection of molecules. To a large degree the access to components is comparable to FnPath, but F–Logic does not consider quite as powerful transformation facilities.

4 Processing XML–Data with FnPath

The *structure of Web–sites* can change over time in a highly dynamic way, even though their semantic contents stays the same. For example, a restructuring of an HTML–page can insert further levels of nesting, such that certain information will be found at a deeper level after the restructuring, or the names of certain elements can be changed.

We have investiagted the Web site of a German company providing stock information: `http://www.finanztreff.de`, and the Web site of the well–known company Amazon: `http://www.amazon.de`. In both cases the goal was to extract some information based on partial (i.e., incomplete) knowledge about the structure of the HTML–pages. We have also applied FnPath to the XML–representation of the knowledge base of the *diagnostic expert system* D3 [9], which can be used for diagnostic tasks in medicine or biology, and for processing *mathematical knowledge* represented in the XML–languages MathMl and OpenMath [7]. In the following we will describe the application of FnPath to the stock example.

Parsing. We have downloaded the HTML–documents from the Web into files. Then we have used the SGML–parser of SWI–Prolog for loading the HTML–documents into Prolog, which we then have transformed to field–notation:

```
original_object(O) :-
   load_html_file(
      'stock_data/02_05_03/dax100.htm', Content),
   xml_swi_to_fn(Content, O).
```

For the HTML–file shown in Appendix A we obtain the field–notation given in Appendix B.

Processing of Stock Data. The following rules can be used for extracting stock data from an HTML–page. The relevant part of the HTML–document with the stock data as well as its representation in field–notation are given in the appendix. The overall structure of the document is unknown, and it could also vary over time. By looking at the HTML–page in the browser it can be seen that the page contains a table, such that each row provides stock information about a stock share, and that one of the rows contains the string "Adidas–Salomon".

The following rules find out where the string "Adidas–Salomon" is located in the document `O`, and they return the corresponding paths shortened by the trailing elements until the first `tr`– or `TR`–element occurs:

```
relevant_path(O, Path_2) :-
   Xs := O^Path_1,
   member('Adidas-Salomon', Xs),
   fn_path_cut_at_end([tr, 'TR'], Path_1, Path_2).
```

This rule is very robust, since it will always find relevant paths if the weak assumptions that we have made are fulfilled. Currently, the only relevant path that is returned is the following:

```
html^body^p^p^p^'Table'^tr^td^'TABLE'^'TBODY'^'TR'
```

The tail `^'TD'^'A'` was cut away by `fn_path_cut_at_end`. Observe, that the tag `'TBODY'` – as well as some other tags – was not present in the original document; it was introduced by the SGML–parser of SWI–Prolog. Surprisingly, the table that we are interested in is located within another table; thus, it would not be enough to just look for a table in the document. All elements that can be reached by a relevant path are extracted, reduced and transformed:

```
result(Line) :-
   original_object(O),
   relevant_path(O, Path),
   Line_1 := O^Path,
   reduce_line(Line_1, Line_2),
   transform_line(Line_2, Line).

reduce_line(Line_1, Line_2) :-
   Element_Patterns = [
      br:_:_, 'BR':_:_ ],
   fn_delete_elements(Element_Patterns, Line_1, Line_2).

transform_line(Line_1, Line_2) :-
   Substitution = [
      X-('FONT':_:[X]), X-(img:_:[X]),
      (td:X)-('TD':_:X), (td:X)-(td:['A':_:X]),
      (th:X)-('TH':_:X) ],
   fn_substitute_elements(Substitution, Line_1, Line_2).
```

We have obtained the previous rules in an iterative design process. As long as the current result contained redundant elements or attributes, these were deleted. Finally, complex elements such as "`'FONT':_:[X]`", "`td:['A':_:X]`", and "`'TD':_:X`" were substituted by simpler ones, namely "`X`" and "`td:X`", respectively. Observe, that the substitutions are applied iteratively in their given sequence; i.e., "`'TD':_:['A':_:X]`" is replaced by "`td:['A':_:X]`" in the first iteration, which is then replaced by "`td:X`" in the second iteration. The obtained results are shown in Appendix C.

The substitution rules were found by iteratively simplifying the structure of the document. Originally, we had started by deleting redundant attributes using the following rule:

```
reduce_line_2(Line_1, Line_2) :-
   Attribute_Patterns = [
      'CLASS':_, rowspan:_, colspan:_,
      'HREF':_, 'SIZE':_, 'FACE':_, 'COLOR':_ ],
   fn_delete_attributes(Attribute_Patterns, Line_1, Line_2).
```

But after deleting enough redundant attributes a condensed result was obtained, which made it possible to come up with the substitution that is used in the predicate `transform_line`; this finally rendered the predicate `reduce_line_2` irrelevant again.

5 Conclusions and Future Work

The present paper has shown that techniques for processing and reasoning about complex Xml–documents can be integrated nicely into Prolog–applications. Thus, it becomes possible to formulate complex queries to Xml–documents in a compact and intuitive way. Instead of embedding an Xml–query language into a procedural language, we propose a homogeneous framework for accessing, transforming and reasoning about Xml–documents in a declarative environment.

The library FnPath has been developed for Xml–documents in field–notation. Currently, we are planning to extend it to alternative document object models for Xml–documents in Prolog, such as the relational representation that is presented in [1], which is based on two relations ref/3 and val/2.

A more detailed description of the built–in predicates of the library FnPath will be given in a separate paper. We will also provide a more formal definition of the semantics of FnPath; in particular, the semantics of tree expressions with variables and evaluable predicates will be defined in a more refined way.

Currently, the predicates of FnPath are evaluated in a top–down and tuple–oriented fashion using Prolog; for computing all answers to a query we use backtracking. For more complicated rule systems containing recursive rules we are planning to use set–oriented bottom–up evaluation as in Datalog.

References

1. *S. Abiteboul, P. Bunemann, D. Suciu:* Data on the Web – From Relations to Semi–Structured Data and Xml, Morgan Kaufmann, 2000.
2. *F. Bancilhon, S. Cluet, C. Delobel:* Query Languages for Object–Oriented Database Systems: the O_2 Proposal, Proc. 2nd Intl. Workshop on Database Programming Languages, 1989.
3. *T. Berners–Lee, J. Hendler, O. Lassila:* The Semantic Web, Scientific American, May 2001.
4. *F. Bry, S. Schaffert:* Pattern Queries for Xml and Semistructured Data. Proc. 17th Workshop Logische Programmierung, 2002.
5. *S. Ceri, G. Gottlob, L. Tanca:* Logic Programming and Databases, Springer, 1990.

6. *M. Gross–Hardt:* Querying Concepts — An Approach to Retrieve XML Data by Means of their Data Types. Proc. 17th Workshop Logische Programmierung, 2002.
7. *B. Heumesser, D. Seipel, U. Güntzer:* Flexible Processing of XML–Based Mathematical Knowledge in a PROLOG–Environment, Proc. Intl. Conf. on Mathematical Knowledge Management MKM'2003, Springer LNCS (to appear).
8. *M. Kifer, G. Lausen:* F–Logic: A Higher–Order Language for Reasoning about Objects, Proc. ACM SIGMOD Conference on Management of Data, 1989.
9. *P. Puppe et al.:* D3. http://d3web.informatik.uni-wuerzburg.de/
10. *D. Seipel:* A PROLOG–Engine for Querying and Transforming XML–Data in Field–Notation, http://www1.informatik.uni-wuerzburg.de/databases/DisLog.
11. *V. Vianu:* A Web Odyssey: from Codd to XML, Proc. Intl. Conference on Principles of Database Systems PODS'2001.
12. XML Query Requirements, Working Draft of the World Wide Web Consortium (W3C), http://www.w3.org/TR/2001/WD-xmlquery-req-20010215.

Appendix A. Stock Data in HTML

The following HTML–table occurs deeply nested within the HTML–document

```
http://www.finanztreff.de/portal/
    kurse_index_listen.htm?u=0&p=0&k=0&symbol=DAX
```

```
<TABLE BORDER=0 CELLPADDING=2 cellspacing=2
       WIDTH=693 BGCOLOR="#c6c6c6">
   <TR>
      <TH CLASS="hl">Bezeichnung<BR>Boerse / WKN</TH>
      <TH CLASS="hl">Letzter<BR>Umsatz</TH>
      <TH CLASS="hl">+/-<BR>%</TH>
      <TH CLASS="hl">GUmsatz<BR>Trend</TH>
      <TH CLASS="hl">Zeit<BR>Datum</TH>
      <TH CLASS="hl">Bid<BR>Ask</TH>
      <TH CLASS="hl">Vortag<BR>Erster</TH>
      <TH CLASS="hl">Hoch<BR>Tief</TH>
      <TH CLASS="hl">Kassa<BR>Volumen</TH>
   </TR>
   <TR>
      <TD CLASS="wl">
         <A HREF="javascript:var ...">Adidas-Salomon</A> <BR>
         XETRA / 500340
      </TD>
      <TD CLASS="wr">
         82,07
         <img src="http://gfx.finanztreff.de/images/tw/1.gif"
              alt="Kurs gegenber Vortag gestiegen"
              width=11 height=10 border=0> <BR>
         0
      </TD>
      <TD CLASS="wr">
         <FONT SIZE="1" FACE="Verdana, Arial, Helvetica"
```

```
               COLOR="#009900"> 1,57 </FONT> <BR>
         <FONT SIZE="1" FACE="Verdana, Arial, Helvetica"
               COLOR="#009900"> 1,95 </FONT>
      </TD>
      <TD CLASS="wr">550.706<BR>=-+=====</TD>
      <TD CLASS="wr">21:15:17<BR>03.05.2002</TD>
      <TD CLASS="wr">0,00<BR>0,00</TD>
      <TD CLASS="wr">80,50<BR>80,00</TD>
      <TD CLASS="wr">82,90<BR>80,00</TD>
      <TD CLASS="wr">0,00<BR>45.196.441</FONT></TD>
   </TR>
   <TR> ... </TR>
   ...
</TABLE>
```

Appendix B. Stock Data in Field–Notation

The following field–notation has been created using the SGML–parser of SWI–Prolog for transforming Html–documents to some Prolog–representation, which we then have transformed to field–notation.

The association list `['CLASS':hl,rowspan:1,colspan:1]`, which occurs many times in the FN–object, is abbreviated by `[***]`.

```
'TABLE':[
   'BORDER':0,'CELLPADDING':2,cellspacing:2,
   'WIDTH':693,'BGCOLOR':'#c6c6c6']:[
   'TBODY':[]:[
      'TR':[]:[
         'TH':['CLASS':hl,rowspan:1,colspan:1]:[
            'Bezeichnung','BR':[clear:none]:[],'Boerse/WKN'],
         'TH':[***]:['Letzter','BR':[clear:none]:[],'Umsatz'],
         'TH':[***]:[+/-,'BR':[clear:none]:[],%],
         'TH':[***]:['GUmsatz','BR':[clear:none]:[],'Trend'],
         'TH':[***]:['Zeit','BR':[clear:none]:[],'Datum'],
         'TH':[***]:['Bid','BR':[clear:none]:[],'Ask'],
         'TH':[***]:['Vortag','BR':[clear:none]:[],'Erster'],
         'TH':[***]:['Hoch','BR':[clear:none]:[],'Tief'],
         'TH':[***]:['Kassa','BR':[clear:none]:[],'Volumen']],
      'TR':[]:[
         'TD':['CLASS':wl,rowspan:1,colspan:1]:[
            'A':['HREF':'...']:[
               'Adidas-Salomon','BR':[clear:none]:[],
               'XETRA/500340']],
         'TD':['CLASS':wr,rowspan:1,colspan:1]:[
            '82,07',
            img:[src:'...',alt:'...',width:11,height:10,border:0]:[
               'BR':[clear:none]:[],0]],
         'TD':[***]:[
```

```
        'FONT':['SIZE':1,'FACE':'Verdana,Arial,Helvetica',
           'COLOR':'#009900']:['1,57'],
        'BR':[clear:none]:[],
        'FONT':['SIZE':1,'FACE':'Verdana,Arial,Helvetica',
           'COLOR':'#009900']:['1,95']],
     'TD':[***]:[550.706,'BR':[clear:none]:[],'=-+====='],
     'TD':[***]:['21:15:17','BR':[clear:none]:[],'03.05.2002'],
     'TD':[***]:['0,00','BR':[clear:none]:[],'0,00'],
     'TD':[***]:['80,50','BR':[clear:none]:[],'80,00'],
     'TD':[***]:['82,90','BR':[clear:none]:[],'80,00'],
     'TD':[***]:['0,00','BR':[clear:none]:[],'45.196.441']],
  'TR':[]:[...],
  ... ] ]
```

Notice that the attributes `rowspan` and `colspan`, which were not present in the original file, were introduced by the SGML–parser.

Appendix C. Pruned Stock Data in Field–Notation

We have obtained the following pruned result facts. The first fact contains the attributes from the header of the original Html–table, and each subsequent fact contains the corresponding data for one stock share.

```
result(
   [ th:[ 'Bezeichnung', 'Boerse / WKN' ],
     th:[ 'Letzter', 'Umsatz' ],
     th:[ '+/-', '%' ],
     th:[ 'GUmsatz', 'Trend' ],
     th:[ 'Zeit', 'Datum' ],
     th:[ 'Bid', 'Ask' ],
     th:[ 'Vortag', 'Erster' ],
     th:[ 'Hoch', 'Tief' ],
     th:[ 'Kassa', 'Volumen' ] ]).
result(
   [ td:[ 'Adidas-Salomon', 'XETRA / 500340' ],
     td:[ '82,07', 0 ],
     td:[ '1,57', '1,95' ],
     td:[ 550.706, '=-+=====' ],
     td:[ '21:15:17', '03.05.2002' ],
     td:[ '0,00', '0,00' ],
     td:[ '80,50', '80,00' ],
     td:[ '82,90', '80,00' ],
     td:[ '0,00', '45.196.441' ] ]).
...
```