An Expert System for the Flexible Processing of XML–Based Mathematical Knowledge in a PROLOG–Environment

Bernd D. Heumesser[1], Dietmar A. Seipel[2], Ulrich Güntzer[1]

[1] University of Tübingen, Wilhelm–Schickard Institute for Computer Science
Sand 13, D – 72076 Tübingen, Germany
{heumesser,guentzer}@informatik.uni-tuebingen.de

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

Abstract. In this paper, we describe techniques for querying and transforming XML–based mathematical knowledge. The XML–documents are transformed into an equivalent PROLOG–structure called *field notation*, which serves as our *Document Object Model* (DOM).
Based on the field notation we provide a powerful and flexible *query language* in a PROLOG–based logic programming environment enabling *intelligent reasoning* about the data. It also offers a method which allows for elegantly encoding transfomations on XML–documents, using a powerful *substitution* mechanism.
We are applying these techniques in an *expert system* for the *classification* and the *retrieval* of ordinary differential equations. The rule–based approach allows to provide a *query* and *transformation language*, which can deal with different kinds of XML–based mathematical documents, such as documents in MATHML and in OPENMATH.

1 Introduction

XML–technology [12] changed the document formats in almost all application domains. The use of XML has led to a separation of presentation and content in many areas. Mathematical knowledge is more and more frequently represented in an XML–based document format. In fact, now the documents can not only contain the mathematical notation for presentation purposes, but can also contain the intended meaning of the described mathematical structure. Representing and storing mathematical achievements and knowledge in a self–describing, extensible and open manner, facilitates Web information systems and simplifies world–wide cooperation among mathematicians.

This knowledge is spread over the Web and it can only be handled if there is a possibility to operate on and to query this information based on the underlying mathematical structure. In the mathematical community there are efforts to make mathematical knowledge more easily usable via the Web. For instance the

Mowgli–project (cf. [1]) works on a development of a technological infrastructure for the creation and the maintenance of a virtual distributed, hypertextual library of mathematical knowledge based on a content description of the information.

Using logic programming techniques for dealing with mathematical knowledge is an approach that has also been used by Dalmas, Gaëtano and Huchet [5], who present a deductive database Mfd2 for mathematical formulas. The information stored in the database can be queried by using a special unification algorithm which takes care of the conditions associated with the formulas. The MBase system of Kohlhase and Franke [8] provides access to different theorem proving systems based on systems using logic programming components. These two systems put the focus on using deduction for theorem proving.

We introduce a Prolog–based environment for handling Xml–based mathematical knowledge. We use two aspects of Prolog: On one hand we apply Prolog–like a procedural programming language to implement a basic library using, e.g., its capabilites to deal with terms and its unification mechanism. On the other hand we use its rule–based features and its inference–engine for deductions based on these libraries. The system is accessed by the Http–Protocol and can retrieve documents stored on the local file system or remote documents adressed by a URL.

The paper is organised as follows: First, we give a brief introduction to the modeling of mathematical knowledge in Xml–languages. In Section 3, we present the Prolog–Document Object Model (DOM) that is used in our expert system; it allows for handling Xml–documents nicely in Prolog. We introduce a query language called FnQuery for accessing and for extracting parts of the documents. We also present a method which allows for elegantly encoding transfomations on Xml–documents, using a powerful *substitution* mechanism. In Section 4, we show how our expert system can deal with different Xml–languages, and we apply these techniques to the domain of ordinary differential equations. Finally, the conclusion gives a summary of our work and a short outlook.

2 Xml–Based Mathematical Knowledge

Xml enables markup languages – so–called applications of Xml – to be defined for a given purpose or for an application domain. Since all of these languages share the same grammatical structure it is possible to have generic tools and software systems to deal with all Xml–applications at least in a general way. Documents in each Xml–application may be stored in an Xml– (or an Xml–enabled) database. Especially the use in the Web is supported by many tools enabling processing Xml–documents for displaying in a standard Web–browser.

At this time the two major Xml–languages for representing mathematical knowledge are MathML and OpenMath. They are *complementary* (cf. [4]). Therefore we are considering both languages in this paper.

2.1 The Content Markup of MathML

The mathematical markup language MathML [14] consists of two parts: One part is the presentation markup used for presentation purposes. In this part of MathML the user specifies how the content should be presented. This may differ from the mathematical structure, because for presentation it is not necessary to specify the exact mathematical structure. The second part of MathML – the content markup – is a language in which mathematical structures may be defined in an exact way.

Since we are intereseted in the mathematical content of documents, we consider only the content markup of MathML. But we have to deal with some presentation markup, too, because the two languages may be mixed, for instance if one uses variables with subscripts in the content markup. In MathML, the set of available Xml–tags is fixed. There are many predefined symbols, functions, and containers. However, it is possible to use other Xml–languages like, e.g., OpenMath inside a MathML–structure.

2.2 The OpenMath–Standard

OpenMath [4] is another standard for representing mathematical knowledge. This language solely deals with the semantic meaning of the knowledge. OpenMath uses an Xml–language for exchanging mathematical objects; in this paper we refer to this representation.

In contrast to MathML, OpenMath defines only a small set of tags. They are just used as containers and the semantic information is given in their attributes. Hence the tag names carry only general information. On the other hand there is a concept to assign semantic information to the objects, namely *content dictionaries*. Content dictionaries hold the meaning of some mathematical symbols. These symbols are used by the `OMS`–tag via addressing the content dictionary with the `cd`–attribute and the symbol with the `name`–tag.

2.3 An Example for MathMl and for OpenMath

Figure 1 shows the representation of the equation $y'(x) + y(x) \cdot \cos(x) = e^{2 \cdot x}$ in the content markup of MathML on the right hand side and the content markup of OpenMath on the left hand side; this equation might appear in a document in another Xml–language, such as, e.g., OMDoc [9] or Xhtml [13]. Observe that – as Figure 1 suggests – the general structure is almost the same in both languages.

In OpenMath the semantic information is encoded in the values of the tags referencing a content dictionary and a symbol name, whereas the MathML–tags are more readable for humans. E.g., the `times`–tag in MathML is denoted in OpenMath as a generic `OMS`–tag with two attributes for addressing the content dictionary and the symbol name.

MathML	OpenMath

<pre><math>
  <apply>
    <eq/>
    <apply>
      <plus/>
      <apply>
        <diff/>
        <bvar>
          <ci>x</ci>
          <degree>
            <cn type="integer">1</cn>
          </degree>
        </bvar>
        <apply>
          <ci type="fn">y</ci>
          <ci>x</ci>
        </apply>
      </apply>
      <apply>
        <times/>
        <apply>
          <ci type="fn">y</ci>
          <ci>x</ci>
        </apply>
        <apply>
          <cos/>
          <ci>x</ci>
        </apply>
      </apply>
    </apply>
    <apply>
      <exp/>
      <apply>
        <times/>
        <cn type="integer">2</cn>
        <ci>x</ci>
      </apply>
    </apply>
  </apply>
</math></pre>
	<pre><OMOBJ>
	  <OMA>
	    <OMS cd="relation1" name="eq"/>
	    <OMA>
	      <OMS cd="arith1" name="plus"/>
	      <OMA>
	        <OMS cd="calculus1" name="diff"/>
	        <OMBIND>
	          <OMS cd="fns1" name="lambda"/>
	          <OMBVAR>
	            <OMV name="x"/>
	          </OMBVAR>
	          <OMA>
	            <OMV name="y"/>
	            <OMV name="x"/>
	          </OMA>
	        </OMBIND>
	      </OMA>
	      <OMA>
	        <OMS cd="arith1" name="times"/>
	          <OMA>
	            <OMV name="y"/>
	            <OMV name="x"/>
	          </OMA>
	        <OMA>
	          <OMS cd="transc1" name="cos"/>
	          <OMV name="x"/>
	        </OMA>
	      </OMA>
	    </OMA>
	    <OMA>
	      <OMS cd="transc1" name="exp"/>
	        <OMA>
	          <OMS cd="arith1" name="times"/>
	          <OMI>2</OMI>
	          <OMV name="x"/>
	        </OMA>
	    </OMA>
	  </OMA>
	</OMOBJ></pre>

Fig. 1. Representation of $y'(x) + y(x) \cdot \cos(x) = e^{2 \cdot x}$

3 Xml and Prolog

For the use inside of our expert system we have developed an *abstract data type* called *field notation* for representing Xml–documents, which supports operations such as accessing children and attributes of Xml–documents or assinging new values to them. It serves as our DOM for building a digital library of fine–grained mathematical objects.

Based on this field notation we provide a powerful and flexible *query language* in a Prolog–based logic programming environment. This bridges the gap between Xml–based digital libraries and deductive (inference–based) databases enabling intelligent reasoning about the data by providing full access to the Prolog programming environment. We have also implemented a *transformation language* that is inspired by Xslt. It can for instance be used for transforming HTML–documents or XHTML–documents – which currently are available on the Web – to well–structured, semantically enriched XML–documents.

3.1 A PROLOG–DOM for XML

We use *association lists* for representing complex objects, cf. [11]. This data structure is familiar to LISP programmers. An association list consists of attribute/value–pairs of the form "$\mathsf{a_i} : \mathsf{v_i}$", where $\mathsf{a_i}$ is an attribute and "$\mathsf{v_i}$" is the associated value. Thus, a complex object O, where $\mathsf{v_i} = \mathsf{O.a_i}$, $1 \leq i \leq n$, can be represented as an association list

$$\mathsf{O} = [\mathsf{a_1} : \mathsf{v_1}, \ldots, \mathsf{a_n} : \mathsf{v_n}].$$

Notice, that the values "$\mathsf{v_i}$" themselves can be association lists (i.e., the definition is inductive) or atomic values. XML–objects can have attributes and sub–elements. Thus, an XML–object with the tag name "A" can be represented as a triple $\mathsf{A} : \mathsf{As} : \mathsf{Es}$, where "As" is an association list for its attribute/value–pairs and "Es" represents its sub–elements. If there are no attributes, then the XML–object can alternatively be represented by $\mathsf{A} : \mathsf{Es}$ instead of $\mathsf{A} : [\,] : \mathsf{Es}$. We call the notation

$$\mathsf{O} = [\mathsf{a_1} : \mathsf{v_1} : \mathsf{w_1}, \ldots, \mathsf{a_n} : \mathsf{v_n} : \mathsf{w_n}] \tag{1}$$

for a list of XML–objects *field notation*. An XML–document can be represented by a list $[\mathsf{A} : \mathsf{As} : \mathsf{Es}]$ containing one element, or simply by that single element if no confusion arises. E.g., the XML–documents of Figure 1 are represented in field notation in Figure 2. Representing complex objects in field–notation rather than as ordinary PROLOG–atoms has got several advantages: Firstly, the sequence of attribute/value–pairs is arbitrary. Secondly, values can be accessed by attributes 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, semi–structured data, such as XML–data, can be represented and queried very elegantly; we will see this in the following.

Examples for FNQUERY. In the following we give some examples for the usage of the field notation and the language FNQUERY. A more detailed description can be found in [10].

For an object `O` of the form (1) in field notation we can select the list `X = w`$_i$ of sub–elements using the statement "`X := O^a`$_i$", and we can select the list `Y = v`$_i$ of attribute/value–pairs using the statement "`Y := O@a`$_i$". Here ":=" is an infix–predicate symbol with two arguments, which evaluates the terms "`O^a`$_i$" and "`O@a`$_i$", and assigns the result to the first argument "X" and "Y", respectively.

```
?- O = [ a:[d:1]:[ b:2, c:3], c:4 ],
   X := O^a,
   Y := O@a.
```

MathML	OpenMath

<pre>math:[]:[
  apply:[]:[
    eq:[]:[],
    apply:[]:[
      plus:[]:[],
      apply:[]:[
        diff:[]:[],
        bvar:[]:[
          ci:[]:[x],
          degree:[]:[
            cn:[type:integer]:[1]]],
        apply:[]:[
          ci:[type:fn]:[y],
          ci:[]:[x]]],
      apply:[]:[
        times:[]:[],
        apply:[]:[
          ci:[type:fn]:[y],
          ci:[]:[x]],
        apply:[]:[
          cos:[]:[],
          ci:[]:[x]]]],
    apply:[]:[
      exp:[]:[],
      apply:[]:[
        times:[]:[],
        cn:[type:integer]:[2],
        ci:[]:[x]]]]]</pre>
	<pre>OMOBJ:[]:[
	  OMA:[]:[
	    OMS:[cd:relation1, name:eq]:[],
	    OMA:[]:[
	      OMS:[cd:arith1, name:plus]:[],
	      OMA:[]:[
	        OMS:[cd:calculus1, name:diff]:[],
	        OMBIND:[]:[
	          OMS:[cd:fns1, name:lambda]:[],
	          OMBVAR:[]:[
	            OMV:[name:x]:[]],
	          OMA:[]:[
	            OMV:[name:y]:[],
	            OMV:[name:x]:[]]]],
	        OMA:[]:[
	          OMS:[cd:arith1, name:times]:[],
	          OMA:[]:[
	            OMV:[name:y]:[],
	            OMV:[name:x]:[]],
	          OMA:[]:[
	            OMS:[cd:transc1, name:cos]:[],
	            OMV:[name:x]:[]]]],
	      OMA:[]:[
	        OMS:[cd:transc1, name:exp]:[],
	        OMA:[]:[
	          OMS:[cd:arith1, name:times]:[],
	          OMI:[]:[2],
	          OMV:[name:x]:[]]]]]</pre>

Fig. 2. Field Notation of the Xml–Documents in Figure 1

```
X = [b:2, c:3]
Y = [d:1]
```

This reminds of the evaluation of an arithmetic expression "`X is 3 * (4+5)`" in Prolog 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".

It is possible to have *complex paths expressions* for selecting sub–elements:

```
?- O = [ a:[d:1]:[ b:2, c:3], c:4 ],
   X := O^a^b.

X = 2
```

In pure Prolog the previous statement would look much more complicated.

A query selecting with multiple path expressions returns a list of objects, namely one object for each selector:

```
?- O = [ a:[ b:2, c:3], c:4 ],
   X := O^[a, c, a^b].

X = [[ b:2, c:3], 4, 2]
```

If a path expression contains variable symbols, then all ground instances of the path expression which are an allowed path in the queried object are generated (on backtracking):

```
?- O = [ a:[ b:2, c:3], c:4 ],
   X := O^a^Path.

X = 2, Path = b ;
X = 3, Path = c

?- O = [ a:[ b:2, c:3], c:4 ],
   2 := O^Path.

Path = a^b
```

Finally, using the operator "*", we can *assign new values* to attributes or elements as follows:

```
?- O = [ a:[d:1]:[ b:2, c:3], c:4 ],
   O_2 := O*[^a@d:5,^c:6],
   O_3 := O^a*[^b:1].

O_2 = [ a:[d:5]:[ b:2, c:3], c:6 ],
O_3 = [ b:1, c:3]
```

3.2 The Transformation Mechanism

To demonstrate the power of the transformation mechanism we show how easily it can be used to convert two different XML–languages. We have built a high level *substitution mechanism* on top of the substitution mechanism provided by PROLOG. These interleaved substitutions allow complex transformations on the documents to be encoded easily.

The following PROLOG–code converts the OPENMATH–document `OM` in field notation from Figure 1 and 2 to a MATHML–document `MM` in field notation:

```
Substitution_Symbols = [
   (eq:[]:[]) - ('OMS':[cd:relation1,name:eq]:[]),
   (plus:[]:[]) - ('OMS':[cd:arith1,name:plus]:[]),
   (times:[]:[]) - ('OMS':[cd:arith1,name:times]:[]),
   (cos:[]:[]) - ('OMS':[cd:transc1,name:cos]:[]),
   (exp:[]:[]) - ('OMS':[cd:transc1,name:exp]:[]),
   (diff:[]:[]) - ('OMS':[cd:calculus1,name:diff]:[]) ],
Substitution_Structure = [
   (apply:[]:[Op,
       bvar:[]:[ci:[]:[Var],
          degree:[]:[cn:[type:integer]:[1]]],X]) -
      ('OMA':[]:[Op,
```

```
            'OMBIND':[]:[
               'OMS':[cd:fns1,name:lambda]:[],
               'OMBVAR':[]:['OMV':[name:Var]:[]],X]]),
    (apply:[]:[ci:[type:fn]:[F]|Es]) -
       ('OMA':[]:['OMV':[name:F]:[]|Es]),
    (ci:[]:[N]) - ('OMV':[name:N]:[]),
    (cn:[type:integer]:[I]) - ('OMI':[]:[I]),
    (apply:[]:Es) - ('OMA':[]:Es),
    (math:[]:Es) - ('OMOBJ':[]:Es) ],
 append( Substitution_Symbols, Substitution_Structure,
    Substitutions ),
 fn_transform_elements(Substitutions,OM,MM).
```

As one can see, the main work that has to be done is to define the substitutions, which are then applied to the object in field notation within the predicate `fn_transform_elements` of our library. Each substitution consists of two parts separated by a minus sign, and it represents a context–sensitive rewriting rule: the right hand expression of a substitution is replaced by the expression on the left hand side.

There are two lists of substitutions: The first one contains the substitutions used for converting the symbols. There are no Prolog–variables in this list because we just need to substitute the OpenMath tags by the corresponding tags in MathML. So we don't need the Prolog–substitution mechanism here. In the second list of substitutions, `Substitution_Structure`, there are some variables and hence the Prolog–substitution mechanism is used.

Now we will show how this works. Consider the following substitution:

```
(apply:[]:[ci:[type:fn]:[F]|Es]) -
   ('OMA':[]:['OMV':[name:F]:[]|Es])
```

The predicate `fn_transform_elements` tries to unify the parenthesized term behind the minus sign. This can only be done, if the content of the `OMA`–tag starts with a variable. This indicates that the variable with the name `F` is a function, which is applied to the other elements `Es` of the content. This construct is translated in MathML by using an `apply`–tag and a `ci`–container for the variable. This variable is characterized by the `type`–attribute as a function. The remaining content `Es` of the `OMA`–tag is left unchanged.

Note that only the substitution for the derivation is a little bit complex – the others are rather straightforward.

Of course we can use the same technology for doing the backwards conversion from MathML to OpenMath. In [3] it is shown how this can be done. We just have to encode the substitutions in the way shown above. The substitution mechanism allows this to be done very easily in a Prolog–based environment like our system.

A transformation from OpenMath to MathML is not always possible, since OpenMath offers the possibility to define new symbols by using new content dictionaries. As the example above shows, a subset of both languages can be identified, which in fact can be transformed.

Another example for data extraction and transformation dealing with HTML–documents can be found in the appendix.

4 Applications in an Expert System

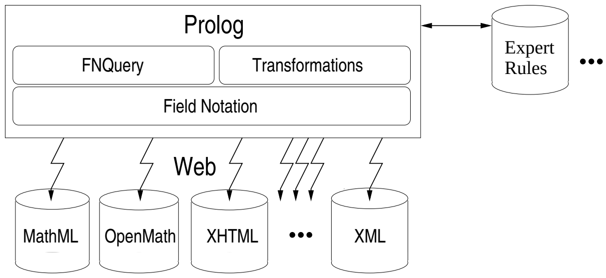

Fig. 3. Architecture of the Expert System

We are currently developing an expert system based on the techniques described in Section 3. It provides different kinds of operations on given documents: The digitally stored and edited mathematical knowledge can be accessed by applying a well–defined and structured access to the types and elements of the XML–documents. Transformations and queries encoded in this syntax are evaluated by the system and they can be mixed with PROLOG–predicates for performing reasoning tasks. I.e., it is possible to couple FNQUERY with the full programming facilities of PROLOG.

The rule–based approach allows to provide a *query* and *transformation language* which can deal with different kinds of XML–based mathematical documents, such as, e.g., documents in MATHML, OPENMATH, or yet to be defined XML–languages for special subject areas.

The following two subsections show how the techniques have been applied. The queries in Section 4.1 demonstrate that our system can handle MATHML– and OPENMATH–documents in a relatively uniform way. In Section 4.2 we apply the system for classifying ordinary differential equations; a more detailed description of this second application is given in [6].

4.1 Application to MATHML and to OPENMATH

We are in the process of extending FNQUERY to an abstract, easy–to–use query language which can be transformed to FNQUERY and then processed by the system to give the user the possibility to define his own queries. The queries generated by the system should cover both MATHML– and OPENMATH–documents.

Because of the different properties of both languages, we have to identify a subset of corresponding structures and concepts which we would like to support.

Let `Object` be an XML–document – represented in field notation – containing some sub–elements in MATHML. Consider the following PROLOG–rule to extract differential equations from the document:

```
query_1(Object,Equations) :-
   findall( Equation_1,
      ( Equation_1 := Object^_^math,
        Equation_2 := Equation_1^apply,
        _ := Equation_2^eq,
        _ := Equation_2^_^diff ),
      Equations ).
```

It extracts a list – possibly with duplicates – of MATHML–objects which represent an equation containing the `diff`–operator. As one can see, this is encoded in just 4 lines of FNQUERY–predicates: The first matches only `math`–tags. The second line ensures that the following tag is an `apply`–tag. Then it is checked that the content of the `apply`–tag (directly) contains an `eq`–tag. At last it is proven that somewhere in this object there is a `diff`–tag.

Note that the above procedure only works for MATHML. But is is quite easy to transform it to deal with OPENMATH. We just have to use the OPENMATH terminology. This results in the following PROLOG–rule to extract all equation objects containing the `diff`–operator from an OPENMATH–document `Object` in field notation:

```
query_2(Object,Equations) :-
   findall( Equation_1,
      ( Equation_1 := Object^_^'OMOBJ',
        Equation_2 := Equation_1^_,
        [relation1,eq] :=
           Equation_1^['OMA'@'OMS'^cd,'OMA'@'OMS'^name],
        [calculus1,diff] :=
           Equation_2^['OMA'@'OMS'^cd,'OMA'@'OMS'^name]),
      Equations ).
```

The predicates look somewhat more complex, but this is only caused by the fact that we have to access the two attributes of the `OMS`–tag and to check both. Since PROLOG–names starting with capital letters are variables, we have to enclose tag names starting with a capital letter with single quotes.

The next step is to combine the predicates used in the above query to obtain predicates working for both languages. In addition, these predicates may easily be extended to deal with other XML–based mathematical markup languages. This is done like shown in the following rules:

```
math_object_contains(Object,diff) :-
   _ := Object^_^apply^diff.
```

```
math_object_contains(Object,diff) :-
   [calculus1,diff] :=
      Object^_^['OMA'@'OMS'^cd,'OMA'@'OMS'^name].
```

These two rules act as alternatives. The predicate succeeds if one of these two rules lead to success. If a set of these predicates is available, then they can be used to implement complex queries. Therefore it is easy to combine this rule–based approach with FnQuery, the transformations mentioned above and the programming facilities of Prolog within our expert system.

4.2 Application to Ordinary Differential Equations

We have applied these techniques to the *classification* and the *retrieval* of ordinary differential equations based on Kamke's rules. Kamke provides an ordered list of ordinary differential equations, cf. Section C of [7]. A user searching for a solution of a given equation has to transform the equation according to a set of rules. The form of the transformed equation determines rather precisely the place of the differential equation within Kamke's list. Thus the user has to browse only a very small list of possible candidates, where he might find his equation in the exact form or a special case of it.

Our expert system supports the search in this collection. We are in the process of defining criteria for the equations to shrink the set of candidates in the list which can match the given equation (in the sense given above) to a reasonable size; for more details see [6]. If this can be done in an effective way, then we can try to unify the given equation with the remaining candidates. This might be done by incorporating efficient *term–rewriting* methods into our system.

5 Conclusions

We access the semantic information encoded in Xml–based mathematical knowledge for providing querying and transformation techniques. Embedded in the Prolog environment, our rule–based system uses the field notation as its Document Object Model, which results in the natural and powerful query language FnQuery, and in a mechanism for the transformation of documents based on substitutions.

This approach features complex reasoning and retrieving tasks which can be implemented in short time in a very powerful language. The rule–based approach allows for a rather generic implementation, which can deal with almost arbitrary Xml–languages that might be used for representing mathematical knowledge. The application of our expert system to the collection of ordinary differential equations shows that this approach is feasible.

Other concepts for transforming and querying Xml–documents are the languages Xslt [16] and XQuery [17]. Both use the language XPath [15] for addressing objects in an Xml–document. Even though they are very powerful languages, our Prolog–based approach has got some advantages, since the embedding of FnQuery in Prolog yields the full access to the potentials of using the Prolog–predicates or self–defined predicates based on the Prolog programming language. Hence, very complex computations and transformations are possible. A more detailed description of the power of FnQuery and a comparison to other query or transformation languages dealing with Xml will be the topic of future research. Our goal is to develop an abstract query language built on top of FnQuery and to implement a graphical, Web–based user interface.

According to Berners–Lee et al. (cf. [2]), the goal of the *Semantic Web* is to add *a logic component for reasoning* about the Web. Therefore embedding techniques for dealing with Xml in a logic programming language like Prolog is a step in this direction.

References

1. A. Asperti, B. Wegner. Mowgli – *A New Approach for the Content Description in Digital Documents.* Proc. of the 9th Intl. Conference on Electronic Resources and the Social Role of Libraries in the Future, Section 4, Volume 1, 2002.
2. T. Berners–Lee, J. Hendler, O. Lassila. *The Semantic Web.* Scientific American, May 2001.
3. D. Carlisle, J. Davenport, M. Dewar, N. Hur, W. Naylor. *Conversion between* MathML *and* OpenMath. Bath/NAG, 2001.
4. O. Caprotti, D. P. Carlisle, A. M. Cohen. *The* OpenMath *Standard.* The OpenMath Esprit Consortium, February 2000.
5. S. Dalmas, M. Gaëtano, C. Huchet. *A Deductive Database for Mathematical Formulas* In: J. Calmet, C. Limongelli (Eds.): Design and Implementation of Symbolic Computation Systems, Springer LNCS, 1996.
6. B. Heumesser, D. Seipel, R. Schimkat, U. Güntzer. *A Web-Information System for Retrieving and Reasoning about* Xml*–Based Mathematical Knowledge* EICM'2002, `http://www-db.informatik.uni-tuebingen.de/forschung/talk_eic.shtml`.
7. E. Kamke. *Differentialgleichungen – Lösungsmethoden und Lösungen.* Akademische Verlagsgesellschaft, 8th Edition, 1967.
8. M. Kohlhase, A. Franke. MBase*: Representing Knowledge and Context for the Integration of Mathematical Software Systems.* Journal of Symbolic Computation, Volume 32, Number 4, 2001.
9. M. Kohlhase. OMDoc*: An Open Markup Format for Mathematical Documents.* Technical Report, Department of Computer Science, Carnegie Mellon University, Pittsburgh, March 2002, `http://www.cs.cmu.edu/~kolhase`.
10. D. Seipel. *Processing* Xml*–Documents in* Prolog. Workshop on Logic Programming WLP'2002.
11. D. Suciu, S. Abiteboul, P. Bunemann. *Data on the Web – From Relations to Semi–Structured Data and* Xml. Morgan Kaufmann, 2000.
12. *Extensible Markup Language (*Xml*) 1.0*, World Wide Web Consortium, October 2000. `http://www.w3.org/TR/2000/REC-xml-20001006`.
13. Xhtml *2.0*, World Wide Web Consortium, August 2002, `http://www.w3.org/TR/2002/WD-xhtml2-20020805/`.
14. *Mathematical Markup Language (*MathML*) Version 2.0*, World Wide Web Consortium, February 2001, `http://www.w3.org/TR/MathML2/`.
15. Xml *Path Language (*XPath*)*, World Wide Web Consortium, November 1999, `http://www.w3.org/TR/xpath`.

16. XSL *Tranformations (*XSLT*)*, World Wide Web Consortium, August 2002, http://www.w3.org/TR/xslt20/.
17. XQUERY *1.0: An* XML *Query Language*, World Wide Web Consortium, August 2002, http://www.w3.org/TR/xquery.

Appendix

We show an example of extracting information from an HTML–document to show the flexibility of the substitution mechanism. Processing HTML–documents is a more difficult task than processing XML–documents. This example shows that our system can even deal with HTML.

Knowledge Extraction from the MathML–Tutorial

The following example has been taken from the MathML–tutorial at

http://www.dessci.com/support/tutorials/mathml/default.stm

which can be reached from http://www.w3.org/Math/. The goal was to extract the examples from the section on *Containers and Operators*. The HTML–document contains sequences – describing examples – of the following form:

```
<p>
   <b>Expression:</b>
      <p>
         <img SRC="misc/set-interval.gif" width="229" height="30">
            <p>
               <b>Markup:</b>
               <pre>
                  <reln> <eq/>
                    <set>
                      <bvar>
                        <ci>x</ci>
                      </bvar>
                      <condition>
                        <reln> <leq/>
                           <cn>0</cn> <ci>x</ci>
                        </reln>
                      </condition>
                    </set>
                    <interval closure='closed-open'>
                      <cn>0</cn> <ci>&infty;</ci>
                    </interval>
                  </reln>
               </pre>
```

The SGML–parser of SWI–PROLOG parses the HTML–document into a PROLOG–atom. When the parser reaches the end of the document, then it automatically closes all open tags. This is important, since the HTML–tags <p> and <img> typically are not closed. Thus, the SGML–parser creates PROLOG–structures, where, e.g., the pre–element is nested within the img–element.

The following PROLOG–predicate extracts the image and the markup for all examples. Firstly, it selects a p–element "X" at any depth, such that "X" contains a b–element with the contents 'Expression:'. Secondly, the attribute list "Img" of the image is selected. Thirdly, the markup "Markup" is selected as the sub–element of "X" that is formatted by "pre". Observe that this sub–element occurs nested within the img–element, since this was not closed in HTML. Since this element was formatted by "pre", it is parsed into a PROLOG–atom "XML_Atom" by the SGML–parser of SWI–PROLOG. This PROLOG–atom can be parsed into field notation using our predicate xml_atom_to_fn_term.

```
fn_term_to_mathml_examples(FN_Term,Examples) :-
   findall( example:[expression:[img:Img:[]],markup:Markup],
      ( X := FN_Term^_^p,
        ['Expression:'] := X^b,
        Img := X^p@img,
        [XML_Atom] := X^p^img^p^pre,
        xml_atom_to_fn_term(XML_Atom,Markup) ),
      Examples ).
```

The following PROLOG–predicate transforms all elements with the tag "SRC" by replacing the tag by "src".

```
fn_mathml_transform_elements(FN_Term_1,FN_Term_2) :-
   Substitutions_1 = [
      (src:Source) - ('SRC':Source) ],
   fn_transform_elements( Substitutions_1,
      FN_Term_1, FN_Term_2 ).
```

Thus we obtain the following XML–element, where the reln–element is the one from the original HTML–document.

```
<example>
   <expression>
      <img src="misc/set-interval.gif" width="229" height="30"/>
   </expression>
   <markup>
      <reln> ... </reln>
   </markup>
</example>
```

Obviously, the original HTML–document could also be simplified by closing all p– and all img–tags. But it turns out that the processing of the modified HTML–document would not be easier for our system.