Querying XML With Lambda Calculi

Pavel Loupal

Dept. of Computer Science, FEE CTU Prague,
Karlovo nám. 13, 121 35
Prague, Czech Republic
loupalp@fel.cvut.cz

Abstract

The aim of this paper is to outline an uniform functional approach useful both for constructing query languages for XML and also for formal description of their semantics. This framework offers an alternative way to today's mainstream query languages – XPath and XQuery. With respect to the goal of the VLDB PhD Workshop it is focused more on the concept than on the complete solution.

1 XML and Query Languages

No doubt XML [4] is nowadays moving the world and thus not surprisingly it might be found almost everywhere. Querying XML data is the essential operation for many algorithms and therefore many researchers focus on this field. Mainstream query languages – XPath [5, 7], used for locating nodes in an XML instance, and XQuery 1.0 [3] (more or less a SQL–like language) – are the results developed by the wide community of contributors.

The XML specification itself does not define any *data model* for XML instances therefore successive query languages had to define a theoretical base for their syntax and semantics. XPath 1.0 is based on the XML Infoset [8], XQuery 1.0 grounds on a newly developed "XQuery 1.0 and XPath 2.0 Data Model" [12]. Based upon this data model its semantics is formally described in [10].

There are various different query languages for XML – especially those used as predecessors of XQuery – e.g. Lorel [1], Quilt [6] or XML-QL [9].

From our perspective one of the most interesting languages is the SXML framework. It seems to be the only framework for manipulating XML in a functional way. The goal of its authors is to implement some of W3C standards – XML, XPath, XSLT – in the Scheme language [11]. SXML takes in account only first versions of these specifications but not the latest drafts. The main components of the project are SSAX, SXML, SXPath, and SXSLT [13, 14].

The key idea is the usage of *S-expressions*. S-expressions are known from the LISP language where there can be either single objects such as numbers, LISP atoms including the special atoms or pairs. The framework claims its fully conformant to XML specification. Regarding to [13] there is a formal algorithm how to rewrite XPath queries to SXPath expressions with identical semantics. SXSLT is apparently an implementation of XSLT 1.0 semantics. Unfortunatelly from published papers it is not exactly obvious that the full XSLT standard was implemented, authors only mention their experiments with real XML documents.

2 Motivation

Evolution of a query language usually brings some changes in its syntax or/and the semantics. For example, the progress from XPath 1.0 to XPath 2.0 (i.e. sub–specification of XQuery 1.0) brought a new data model for XML. It was a logical and necessary step forced by requirements given for the new language. Our idea is to use such formalism (formal system) as a base for the language that is *universal* enough to avoid big changes when enriching the language with new constructs. Therefore the idea of a functional approach combined together with basic lambda calculi operations (lambda abstractions and lambda applications) seems to be very simple and not restrictive, rather it allows to be easily extended in the future.

Our work tries to link to the idea of the XML-λ Framework (see Section 3), explore it further and study its various aspects from different points of view. Its author already identified some issues and drawbacks of the framework. We see one of the biggest issues the missing comparison of the proposed query language with the XQuery language. The question of expressive power of these two languages is very chal-

Proceedings of the VLDB2006 Ph.D. Workshop, Seoul, Korea, 2006

lenging because these approaches represent different formal systems and seems to be not solved yet.

To lay out more specific goals for our consecutive work we formulate the following aims: (a) Utilize the XML-λ Framework for formal description of XQuery semantics, (b) Construct a XQuery-to-XML-λ query compiler (and vice–versa) and compare expressive power of XQuery and XML-λ query languages, (c) Evaluate functional approach for various aspects related to querying XML.

3 The XML-λ Framework

XML-λ framework is an idea of using functional approach together with lambda calculi [2] and utilize it for querying XML data. The proposal was originally developed by Pokorný [16, 17].

The following sections outline the theoretical base of the XML-λ Framework and the query language built upon it. In Section 3.3 we show an example of a query evaluation.

3.1 Concept

The type system is built on base $\mathcal{B}$ – a set containing a finite number of base types $S_1, \ldots, S_k$ $(k \geq 1)$. Type hierarchy is then created by following inductive definition:

Type System T. Let $\mathcal{B}$ is a set of primitive types $S_1 \ldots S_n, n \geq 1$. *Type System T* over base $\mathcal{B}$ is the least set containing types given by 1.-4.

1. *base type:* each member of $\mathcal{B}$ is type over $\mathcal{B}$
2. *functional type:* if T_1 and T_2 are types over $\mathcal{B}$, then $(T_1 \rightarrow T_2)$ is also a type over $\mathcal{B}$
3. *n-tuple type*: if $T_1, \ldots, T_n$ $(n \geq 1)$ are types over $\mathcal{B}$, then $(T_1, \ldots, T_n)$ is type over $\mathcal{B}$
4. *union type:* if $T_1, \ldots, T_n$ $(n \geq 1)$ are types over $\mathcal{B}$, then $(T_1 + \ldots + T_n)$ is type over $\mathcal{B}$

Subsequently we define a regular type system T_{reg} that extends type system T with regular constructs:

Type System T_{reg}. Let $\mathcal{B}$= $\{String, Bool\}$, let $NAME$ be a set of names. Type System T_{reg} is the least set containing types given by 1.-6.

1. Every member of the base $\mathcal{B}$ is an (*primitive*) type over $\mathcal{B}$.
2. *named character data:* Let $tag \in NAME$. Then $tag : String$ is an (*elementary*) type over $\mathcal{B}$, $tag :$ is an (*empty elementary*) type over $\mathcal{B}$.
3. Let T be a group type or named character data. Then
   - *zero or more:* $T*$ is a type over $\mathcal{B}$.
   - *one or more:* $T+$ is a type over $\mathcal{B}$.
   - *zero or one:* $T?$ is a type over $\mathcal{B}$.
4. *alternative:* Let T_1 and T_2 be types. Then $(T_1|T_2)$ is a type over $\mathcal{B}$.
5. *sequence:* Let $T_1, \ldots, T_n$ be types. Then $(T_1, \ldots, T_n)$ is a type over $\mathcal{B}$.
6. *named type:* Let T be a type given by a step from 3.-5. Let $tag \in NAME$. Then $tag : T$ is a type of $\mathcal{B}$.

Note here that this type system T_{reg} is still suitable to describe types not only in a concrete data structure (in our case tree structure) but for all data that might be tagged with name - for example data accordant to the relational model.

Having the T_{reg} type system we have to extend it to be able to work with XML data. We build the type system T_E induced by T_{reg}. Key idea is to define *abstract elements* that are particular XML elements with some content and also define a set containing all abstract elements within an XML instance – E .

Type System T_E. Let T_{reg} over base $\mathcal{B}$ be a type system from definition 3.1 and E is the set of abstract elements. Then *type system T_E* induced by T_{reg} is the least set containing type given by this rule:

- Let $tag : T \in T_{reg}$. Then $TAG : T$ is a member of T_E. (Replacement of all tags in $tag : T$ by upper case version)

Note one remark: we can see the type system T_E as function container that adds functions for extracting data values from elements (via abstractions and projections) with two ways

- *simple element:* if $tag : String \in T_{reg}$, then $(E \rightarrow tag : String) \in T_E$
- *compound element:* if $tag : T \in T_{reg}$, then $(E \rightarrow T') \in T_E$

3.2 XML-λ Query Language

A typical query has a main (body) part – an expression to be evaluated over data – and a constructor part that wraps query result and forms the XML output. XML-λ's Query Language (XLQL) is based on λ-terms defined over the type system T_E.

Main constructs of the language are *variables*, *constants*, *tuples*, use of *projections* and λ-calculus operations – *abstractions* and *applications*. Tagged terms might be used for declaring functions. Syntax of this language is similar to λ-calculus expression thus queries are structured as λ-expresions, i.e. $\lambda \ldots (\lambda \ldots (expression) \ldots)$. In addition, there are also typical constructs such as logical connectives, constants or comparison predicates.

XML-λ Query Language is inductively defined as the least set containing all terms created by application of following rules:

Let $T, T_1, \ldots, T_n, n \geq 1$ be members of T. Then

1. *variable:* each variable of type T is a term of type T
2. *constant:* each constant (member of $\mathcal{F}$) of type T is a term of type T
3. *application:* if M is a term of type $((T_1, \ldots, T_n) \rightarrow T)$ and $N_1, \ldots, N_n$ are (in the same order) types $T_1, \ldots, T_n$, then $M(N_1, \ldots, N_n)$ is a term of type T
4. *λ-abstraction:* if $x_1, \ldots, x_n$ are distinct variables of types $T_1, \ldots, T_n$ and M is a term of type T, then $\lambda x_1, \ldots, x_n(M)$ is a term of type $((T_1, \ldots, T_n) \rightarrow T)$
5. *n-tuple:* if $N_1, \ldots, N_n$ are terms of types $T_1, \ldots, T_n$, then $(N_1, \ldots, N_n)$ is a term of type $(T_1, \ldots, T_n)$
6. *projection:* if $(N_1, \ldots, N_n)$ is a term of type $(T_1, \ldots, T_n)$, then $N_1, \ldots, N_n$ are terms of types $T_1, \ldots, T_n$
7. *tagged term:* if N is a term of type $NAME$ and M is a term of type T then $N : T$ is a term of type $(E \rightarrow T)$.

Simple implementation of this language is available in [18].

3.3 Example of a Query

Following example should give more detailed view to an evaluation of a XML-λ query. Let us consider the DTD and a conforming XML instance as shown in Figure 1 and Figure 2 and a query that will return first names of all authors mentioned in the document.

```
<!ELEMENT book (author+,
                title, price)>
<!ELEMENT author (first, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Figure 1: A Document Type Definition Example

Solution First we specify a set E of all abstract elements contained in XML instance as follows (note that subscripts are here only to distinguish abstract elements of the same type):
$E = \{book, author_1, first_1, last_1, author_2, first_2, last_2, name, price\}$

```
<book>
  <author>
    <first> Jaroslav </first>
    <last> Pokorny </last>
  </author>
  <author>
    <first> Karel </first>
    <last> Richta </last>
  </author>
  <title> XML Technology </name>
  <price> 155.00 </price>
</book>
```

Figure 2: An XML instance

Upon this base we construct type system T_E of functions based by given XML schema (in our case DTD). See the list of types available:

$BOOK : (AUTHOR+, TITLE, PRICE)$,
$AUTHOR : (FIRST, LAST)$,
$FIRST : String$,
$LAST : String$,
$TITLE : String$,
$PRICE : String$

Now, let us have a look at the following simple query solving our example:

$$\lambda f \ \ (/book/author(a) \quad and \quad a/first = f)$$

Evaluation of this query takes place in following way: this query is composed of one high-level λ-term and contains two variables called a and f. Navigation over the structure of elements is implemented as continuous λ-applications combined with projections (performed by element name). When evaluating the value of the query first the value of a is evaluated (note that $/book/author(a)$ is syntactically equivalent to $author(book(a))$). This term is evaluated using sequent application of *book* and *author* functions. Then the variable a contains an abstract element of type *author*. We apply to this element function *author* that returns a n-tuple of elements from E. Last step is projection from this n-tuple into the `first` component. The result of this operation is a particular abstract element.

Hereby we receive the output (without enveloping elements): `Jaroslav, Karel`.

3.4 Utilizing the Functional Approach

In the previous sections we describe a query language based on the XML-λ Framework. It uses *functions* as a data model for XML documents and lambda calculi abstractions and applications for development of a query language suitable for querying XML.

Beside that we can also use this data model for different purposes. Due to universality of the λ-calculus,

we can construct a language for modifying XML data. Generally it means performing changes (additions/removals) in the set E of abstract elements for a particular XML instance and eventual changes in the type system T_E. With this approach we can also be able to express integrity constraints for XML data. This research direction however lays out of scope of this paper.

4 Relationship Between XQuery and XML-λ

XQuery [3] is a mainstream language for querying XML. The language uses expressions as its fundamental constructs. Basic expressions are literals and variables, for navigating in XML documents it uses XPath expressions. Related specification [15] contains an extensive list of predefined numeric, arithmetic, string or date-time functions.

One important type of expression are the *FLWOR expressions*. FLWOR is a shortcut for For, Let, Where, Order-By, Return statements that are used in the same way as in imperative languages. This type of expression is sometimes compared to SQL SELECT statement known from relational databases.

The goal of our research is to explore the relationship between XQuery and XML-λ. Thus, in XQuery we can write a query returning all first names in our bibliography database in following way:

```
<result> {
  for $x in doc("bib.xml")/book/author
    return (
      <fname>
        {$x/first/text()}
      </fname>)
} </result>
```

A query with the same semantics but written in XML-λ has (of course) different syntax as shown below:

```
xmldata("bib.xml")
lambda fname f
  (/book/author(a) and a/first = f)
```

Both queries however return the same result:

```
<result>
  <fname> Jaroslav </fname>
  <fname> Karel </fname>
</result>
```

We want to show that all queries in XQuery can be rewritten into XML-λ and vice–versa. This claim should be backed by formal description of denotational and operational semantics of both languages and by finding a transformation between semantic subtrees of all language constructs.

5 Future Work

Future work should lead into a successful completion of the dissertation thesis. Therefore we need to define concrete tasks and formulate consecutive experiments.

The proposed dissertation thesis investigates an utilization of the λ-calculus based framework for description of query language semantics. We plan to formulate both denotational and structured operational semantics of the XQuery language. Having this formalism available we will compare the expressive power of these languages and further study their properties. Our primary aim is to define the existing semantics of XQuery in a functional way as a base for ongoing experiments and then compare their expressive power by finding transformations between queries in XQuery and XML-λ.

There are two concrete ongoing tasks – specification of XQuery semantics by using the XML-λ Framework and consecutive construction of transformation engine for converting XQuery and XML-λ queries.

5.1 Specification of XQuery Semantics Using λ-calculus

The functional approach applied in XML-λ might be used not only for constructing a query language but also for a different purpose – description of semantics of various query languages. Within our research we want to describe the semantics of the mainstream XML query language XQuery with help of the functional approach based on λ-calculus – XML-λ. Taking into account the type system used we expect the semantics to be easier to express in comparison with existing W3C semantics.

With respect to the scope of the XQuery semantics specification this topic represents a complex theoretical research.

5.2 Bi-directional Transformation of XQuery and XML-λ Queries

Expression of the XQuery semantics with help of λ-calculus gives us an opportunity to compare expressive power of XQuery and XML-λ. Because of the fact that the semantics of XML-λ query language is closely related to its syntax we can propose a method for conversion of queries based on language's semantics. The result of this aim is to have a transformation engine and its implementation for converting XQuery queries in XML-λ and vice–versa available. The conversion process is generally a transformation of an input semantic tree in one language to an output semantic tree in the other language.

6 Conclusion

This paper outlines a framework for querying XML data based on a functional approach combined with lambda calculi operations. The functional approach

is not common nowadays but offers a universal way that might be used as a base for constructing various query languages for XML. Beside that it can be also used for specifying semantics of these languages. Having the semantics formalized with the same tool we can compare their expressive power and evaluate their relationship.

For consecutive work to lead into a dissertation thesis we plan to describe semantics of XQuery using the functional framework. With this formalism we can construct a XQuery-to-XML-λ compiler (and vice–versa) by transformation of respective semantic trees. These transformations describe relationships between various language constructs. With knowledge of these relationships we can then compare XQuery's expressive power with a query language based on lambda calculi.

The main contribution of the thesis should be seen in using functional approach for description of XQuery semantics and for evaluation of properties of lambda calculi based framework and its suitability for querying XML.

7 Acknowledgements

This research has been partially supported by MŠMT under research program MSM 6840770014.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, pages 68–88, 1997.

[2] H. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science, Volumes 1 (Background: Mathematical Structures) and 2 (Background: Computational Structures), Abramsky & Gabbay & Maibaum (Eds.), Clarendon*, volume 2. 1992.

[3] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language, September 2005. http://www.w3.org/TR/xquery/.

[4] T. Bray, F. Yergeau, J. Cowan, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language (XML) 1.1, February 2004. http://www.w3.org/TR/2004/REC-xml11-20040204/.

[5] D. Chamberlin, A. Berglund, and e. a. Scott Boag. XML Path Language (XPath) 2.0, September 2005. http://www.w3.org/TR/xpath20/.

[6] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML query language for heterogeneous data sources. *WebDB, Lecture Notes in Computer Science, volume 1997*.

[7] J. Clark and S. DeRose. XML Path Language (XPath) 1.0, November 1999. http://www.w3.org/TR/xpath.

[8] J. Cowan and R. Tobin. XML information set (second edition), April 2004. http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.

[9] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. XML–QL: A query language for XML. *The Query Language Workshop (QL), Cambridge, MA*, 1998.

[10] D. Draper, P. Fankhauser, M. Fernández, A. Malhotra, K. Rose, M. Rys, J. Siméon, and P. Wadler. XQuery 1.0 and XPath 2.0 formal semantics, September 2005. http://www.w3.org/TR/xquery-semantics/.

[11] R. K. Dybvig. *The Scheme Programming Language*. The MIT Press, 2003. Available online at http://www.scheme.com/tspl3/.

[12] M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 Data Model, September 2005. http://www.w3.org/TR/xpath-datamodel/.

[13] O. Kiselyov. SXML specification, 2004. http://okmij.org/ftp/Scheme/SXML.html.

[14] K. Lisovsky. SXPath: XPath made functional. In *International Lisp Conference ILC 2003*, New York, US, October 2003.

[15] A. Malhotra, J. Melton, and N. Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators, September 2005. http://www.w3.org/TR/xpath-functions/.

[16] J. Pokorný. XML functionally. In B. C. Desai, Y. Kioki, and M. Toyama, editors, *Proceedings of IDEAS2000*, pages 266–274. IEEE Comp. Society, 2000.

[17] J. Pokorný. XML-λ: an extendible framework for manipulating XML data. In *Proceedings of BIS 2002*, Poznan, 2002.

[18] P. Šárek. Implementation of the XML lambda language. Master's thesis, Dept. of Software Engineering, Charles University, Prague, 2002.