XPath 2.0: It Can Sort!

Pavel Hloušek
Dept. of Software Engineering
Phaculty of Mathematics and Physics
Charles University, Prague
pavel.hlousek@mff.cuni.cz

ABSTRACT

As XML finds its place in information technology, query languages for XML attract much attention. Among them XPath is the most known. This article reveals a surprising fact that the upcoming XPath 2.0 is capable of sorting sequences, which is proved formally. We also mark out an incompleteness of XQuery Core and an insufficiency of definition of XQuery Core sorting semantics. The fact that XPath 2.0 can sort arbitrary sequences is used to fix it.

1. INTRODUCTION

In recent years, XML has been widely adopted by IT industry, mainly for its suitability for data exchange in heterogeneous systems. However, this is not the only way XML is used. There is a number of projects that try to store XML documents in their native form. Both these XML applications need a way to query XML data.

A lot of languages has been developed to query XML, some of them inspired by query languages for semistructured data like Lorel [1], Quilt [5], XML-QL [8], some of them written from scratch like XSLT [6]. Now, the most recognized XML query language is XPath.

XPath [7] first appeared as a part of XSLT to match patterns and find nodes in an XML document. Nowadays, W3C is almost done with development of XPath 2.0, which is a part of an effort to provide XPath 2.0, XQuery 1.0, and XSLT 2.0 with the same underlying strongly typed data model.

In this paper, we show that though the upcoming XPath 2.0 has no explicit syntax to express sorting, it can sort arbitrary sequences, which we find surprising. The only constraint is that a sorting condition, i.e. whether one item of a sequence is less than another item of this sequence, is expressible in XPath 2.0 and that this condition represents a partial order on items of the sequence being sorted.

As XPath 2.0 is a subset of XQuery 1.0, we get interesting consequences in XQuery. We identify an incompleteness of XQuery Core and also an insufficiency of formal semantics definition of sorting in XQuery and show how our sorting expression can be used to fix it.

Another consequence of XPath 2.0's capability of sorting is that XQuery really does not add too much to XPath 2.0. If we think it over, it is only the ability to define functions and construct new nodes. Everything else can be realized in XPath.

Contributions. The main contribution of this paper is the formal proof of XPath 2.0's capability of sorting arbitrary sequences. The following are the main features of our approach to sorting in XPath.

- *Generality.* Any sequence of items can be sorted.
- *Complex ordering.* The ordering expression can be any XPath 2.0 expression, thus orderings like "order by last name and within equal last names order by first name" are possible.
- *Inefficiency.* By extending XPath with range expressions like `(1 to count(//cd))`, and for-return expressions, the language became capable of sorting. But an extra cost has to be paid: it is awfully inefficient with time complexity $O(n^3)$ in a number of items in the sorted sequence. A question arises, if some form of an order by expression should be added to the language to achieve better performance of XPath processors.

As a consequence of sorting in XPath, it is possible to express sorting in XQuery while not breaking the consistency of XQuery Core, unlike the nowadays W3C specification of XQuery formal semantics. More on this in section 7.

Structure. This article is organized as follows. First in section 3, we introduce a data model that we use. Then in section 4, we introduce selected features of XPath 2.0 that distinguish it from its 1.0 predecessor. Sorting in XPath is first introduced on an example in section 5, which is then formalized and proved to work correctly in section 6. Finally in section 7, we explain that inconsistency in sorting semantics of XQuery can be fixed using our sorting expression. The conclusion is provided in section 8.

2. RELATED WORK

Since we do not know about any similar work on this topic, we rather briefly introduce the family of W3C specifications in this section.

First of all, XML itself is defined in [4]. The common data model is described in [10] with references to XML Schema typing defined in [4]. XPath 2.0 itself is defined in [2] and XQuery 1.0 in [3]. Both XPath and XQuery share a common set of functions and operators defined in [12]. For XPath and XQuery formal semantics is defined in [9].

All of these W3C specifications except XML itself are so called Last Call Working Drafts, which means that changes may appear later in them.

3. DATA MODEL

The family of W3C upcoming XML query languages XPath 2.0, XQuery 1.0, and XSLT 2.0 has been designed to share the same underlying data model [10]. The emphasis is put on strong typing, which is not what we are interested about.

In this article, we use the untyped core of this model. The basic structure is a *sequence*. A sequence is an ordered sequence of *items*, which are either *node references* or *atomic values*. With $\langle S, \prec_S \rangle$ we denote a sequence where S is a set of items and $\prec_S \subseteq S \times S$ is a total order of items in S.

In fact, when order of items in a sequence is defined this way, we have to extend this data model such that all items have an identity, even atomic values. So, sequences like (2,1,2) can be described in the model. However, this sort of item identity is needed only to formally express order of items and is not referred to by XPath expressions. Notice, that item identity is different from node identity, which is not affected and which can be referred to by XPath expressions.

4. XPATH 2.0

In this section, we briefly introduce XPath 2.0 focusing on selected enhancements of the language as compared to XPath 1.0.

XPath 2.0 [2] is a compact language with non-XML syntax, whose main purpose is to identify parts of an XML document. It is developed by the W3 Consortium as a part of an effort to provide XPath, XQuery, XSLT, and other specifications with the same underlying strongly typed data model. XPath 2.0 evolved as an extension of a successful XPath 1.0 [7], which has been a W3C recommendation since 1999.

As of writing this article, XPath 2.0 specification is still a W3C Working Draft, which means that changes to the language may appear later. We refer to XPath 2.0 specification dated 11th February 2005.

We identify several features of XPath 2.0 that distinguish it from its 1.0 predecessor and that are important to a reader who wants to understand sorting capabilities of the language.

Sequences. The prior version of XPath has been designed to return a *set of nodes* often referred to as a *nodeset*. Thus, no order could have been defined on the result. This has changed in XPath 2.0 that is designed to return an ordered *sequence* of items, where an item is either a node or an atomic value.

Sequence constructors. While XPath 1.0 has no expression to construct data that do not exist in a queried document, XPath 2.0 introduces an expression to construct a sequence. For example, `("hello", "world")` is a sequence of two string values: hello, and world. Further, it introduces so called *range expression* that constructs a sequence of integers. For example, `(1 to 3)` expression constructs the same sequence as `(1, 2, 3)` expression. An important property of a range expression is the fact that an expression can appear in it, e.g. `(1 to count(//cd))` constructs a sequence of integers 1, 2, ..., number of CDs in a document.

Sequence iterator. XPath 2.0 introduces for-in-return expression that iterates over a sequence, e.g. `for $i in //cd return $i/title` returns a title for each CD in a document. The semantics of the for-in-return expression is that a return expression is evaluated once for each variable binding, where a variable is bound to each item in a sequence returned by the in expression.

Conditional expression. XPath 2.0 defines an if-then-else expression with usual meaning. For example, `if (0 = count(//cd)) then 0 else 1` returns 0 if there is no CD element in a document, otherwise it returns 1.

In the following, we use function count() that given a sequence returns a number of items in that sequence, and function empty() that given a sequence returns true only if the sequence has no items. Both these functions are defined among XPath and XQuery standard functions in [12].

5. BY EXAMPLE

Here, we present a general XPath expression that sorts items in a sequence represented by an XPath expression S. The whole expression is for clarity divided into three separate functions that we call my:is-less-than(), my:count-less-than(), and my:sort() that can be assembled together to form a single XPath expression. The three functions have the following meaning.

First, function my:is-less-than($x, $y) represents a less-than relation on items from S. It returns true iff $x is less than $y, whatever to be less than means. In the following, we simply use the usual comparison operator `<`, but more complex expressions can be provided as shown later.

```
my:is-less-than($x, $y) {
  $x < $y
}
```

Second, function my:count-less-than($x, S) returns a number of nodes in a sequence S that are less than $x by means of the less-than relation defined by the my:is-less-than() function. The for loop iterates over S creating so a sequence of items from S that are less than $x. Function count() is applied to this sequence to count a number of items that are less than $x in S.

```
my:count-less-than($x,S) {
  count(
    for $y in S
    return
      if (my:is-less-than($y, $x))
      then $y
      else ()
  )
}
```

Finally, function my:sort(S) returns the sorted sequence.

```
my:sort(S) {
  for $i in (0 to count(S) - 1)
  return
    for $x in S
    return
      if ($i = my:count-less-than($x, S))
      then $x
      else ()
}
```

Let's look how it works. We examine the outer loop first. In its first iteration, it returns each item x in S for which there exists *no* item y in S such that y is less than x – the minimum of S. In its second iteration, it returns each item x in S for which there is *exactly one* node y in S that is less than x. In its third iteration, exactly two, etc. In its final iteration, it returns each item x in S for which all other items in S are less than x – the maximum of S.

If there is a subset of nodes that are equal in S by means of the less-than relation then they will all be returned in a single iteration of the outer loop. In such a case, some iterations return nothing. The inner loop guarantees that the initial sequence order of the equal nodes is preserved in the resulting sequence.

The following XPath 2.0 expression is an example of an assembled expression that returns "sorted CD titles".

```
for $i in (0 to count(//cd/title) - 1)
return
  for $x in //cd/title
  return
    if ($i = count(
          for $y in //cd/title
          return
            if ($y < $x)
            then $y
            else ()
        )
    )
    then $x
    else ()
```

Earlier, we mentioned that it is possible to express more complex sort order by modifying the my:is-less-than() function. Since this is the only place where the semantics of the less-than relation is defined, it is also the only place were it needs to be changed.

For example, we want to order a sequence by authors first by their last and second by their first name. We change the my:is-less-than() function to the following.

```
my:is-less-than($x, $y) {
  ($x/last < $y/last) or
  ($x/last = $y/last and $x/first < $y/first)
}
```

One more example. The following XPath 2.0 expression returns "CD titles ordered by CD authors first by their last name and second by their first name". Notice, that this is a special case of the above that sorts according to information that lies outside the subtree of the sorted nodes.

```
for $i in (0 to count(//cd/title) - 1)
return
  for $x in //cd/title
  return
    if ($i = count(
          for $y in //cd/title
          return
            if ( ( $y/parent::cd/author/last <
                   $x/parent::cd/author/last )
                 or
                 ( ( $y/parent::cd/author/last =
                     $x/parent::cd/author/last )
                   and
                   ( $y/parent::cd/author/first <
                     $x/parent::cd/author/first ) )
            then $y
            else ()
        )
    )
    then $x
    else ()
```

6. FORMALLY

In this section, we formally define expression *Sort* and prove that it sorts each given sequence. Notice, that we define sorting with respect to a given partial order, which is handy later as XQuery defines its ordering semantics on the partial order basis.

Throughout this section, we consider only such partial order relations whose characteristic functions are expressible with some XPath expression.

First, we define equivalence $=_R$ on items in partial order R.

DEFINITION 1. *Let R be a partial order. With $=_R$ we denote a set of all items that are equal in R.*

$$x =_R y \text{ iff } R(x,y)\&R(y,x)$$

Next, we define the less-than relation with respect to a given partial order R by removing equivalence from R. This is needed, since we do not want to count items that are less than or equal to the current item, which is the meaning of R, but rather we want to count items that are sharply less than the current item.

DEFINITION 2. *Let R be a partial order. With $<_R$ we denote a total order $R \setminus =_R$.*

We should note, that if a characteristic function of R is expressible with an XPath expression then even $=_R$ and

$<_R$ are expressible with an XPath expression using boolean operators. We should also note, that the my:is-less-than() function defined in the previous section is an example of a characteristic function of $<_R$ relation.

LEMMA 1. *Let S be a set, and $R \subseteq S \times S$ be a partial order on S. Then for each $x, y \in S$ either $x =_R y$, or $x <_R y$, or $y <_R x$.*

Proof. This comes from the relations between partial order R, equivalence $=_R$, and total order $<_R$, namely $R == _R \cup <_R$ and $=_R \cap <_R = \emptyset$. □

The following formally defines the natural notion of number of items in a sequence that are less than the given item, where *CLT* stands for count less than.

DEFINITION 3. *For S a set, $R \subseteq S \times S$ a partial order on S, and x an item, we define function $CLT_R(x, S)$.*

$$CLT_R(x, S) =_{\text{def}} |\{\ y \in S \mid y <_R x\ \}|$$

Next, we define the *CountLessThan* XPath expression with respect to the given partial order.

DEFINITION 4. *Let S be a set, and $R \subseteq S \times S$ be a partial order on S. With expression $CountLessThan_R(x, S)$ we denote the following XPath expression.*

```
count(
  for $y in S
  return
    if ($y <_R x)
    then $y
    else ()
)
```

The following lemma says that the $CountLessThan_R(x, S)$ expression really counts items from S that are sharply less than x in terms of R. We provide the proof in full detail in [11].

LEMMA 2. *Let S be a set, and $R \subseteq S \times S$ be a partial order on S. Then the result of $CountLessThan_R(x, S)$ is equal to $CLT_R(x, S)$, which is a number of items in S that are less than x in terms of $<_R$.*

The following lemma provides bounds to $CLT_R(x, S)$ and due to Lemma 2 to results of $CountLessThan_R(x, S)$ as well.

LEMMA 3. *Let S be a nonempty set, and $R \subseteq S \times S$ be a partial order on S. Then for each $x \in S$ the following holds.*

$$0 \leq CLT_R(x, S) \leq |S| - 1$$

Proof. The lower bound is equal to zero, as the cardinality of a set cannot be less than zero.

The upper bound cannot be more than $|S|$, which is guaranteed by the first condition $y \in S$ in the definition of CLT_R. Moreover, it cannot be more than $|S| - 1$, which is guaranteed by the second condition $x <_R y$. As $<_R$ is irreflexive, for each $x \in S$ at least x itself is not present in $\{\ y \in S \mid y <_R x\ \}$. □

Notice, that the lower bound is reached for *all* minimal items in S, i.e. items for which there is no less-than item in S, and that there can be more of them equal to each other in terms of $=_R$. Conversely, the upper bound is reached only for the maximum item in S, which has not to exist if there are multiple maximal items. If there are multiple maximal items then they are again equal to each other in terms of $=_R$.

The following lemma claims that equal items have equal counts of less-than items.

LEMMA 4. *Let S be a set, and $R \subseteq S \times S$ be a partial order on S. Then for each $x, y \in S$ such that $x =_R y$ the following holds.*

$$CLT_R(x, S) = CLT_R(y, S)$$

Proof. This lemma can be rewritten as follows.

$$x =_R y \text{ implies } (\forall z \in S : z <_R x \text{ implies } z <_R y)$$

For contradiction, suppose that the above is not true, so suppose that such $z \in S$ exists, for which $z <_R x$ and not $z <_R y$. Combining this fact and Lemma 1, we get that either i) $y <_R z$, or ii) $z =_R y$ holds.

The following inequalities are contradictions, which comes from the fact that $<_R$ is a total order, and $=_R$ is an equivalence.

Ad i) $z <_R x\ =_R\ y <_R z$.

Ad ii) $z =_R x\ =_R\ y <_R z$. □

Next, let's define for a sequence the property of being sorted.

DEFINITION 5. *Let $\langle S, \prec_S \rangle$ be a sequence, and $R \subseteq S \times S$ be a partial order on S. We say that a sequence $\langle S', \prec_{S'} \rangle$ is* a sequence $\langle S, \prec_S \rangle$ sorted with *R iff for each $x, y \in S$ the following conditions hold.*

i) $S' = S$

ii) $x <_R y$ implies $x \prec_{S'} y$

iii) $x =_R y$ implies $x \prec_S y$ implies $x \prec_{S'} y$

We refer to a sequence $\langle S, \prec_S \rangle$ as the initial sequence and a sequence $\langle S', \prec_{S'} \rangle$ as the sorted sequence.

The first condition of the above definition requires the sorted sequence to comprise exactly the same items as the initial sequence.

The second condition requires that if x is less than y in terms of the sorting order R, then x has to precede y also in the sorted sequence.

The third condition requires that if two items are equal in R then the order of x, and y in the initial sequence has to be preserved in the sorted sequence.

Notice, that we define a sequence to be sorted with respect to an initial sequence. Thus, an initial sequence has to exist prior to sorting it.

The above definition of a sequence being sorted is based on the semantics of XQuery's *stable sorting*, which sticks to the idea of preserving the order of the initial sequence. We make use of this to fix the formal semantics of XQuery sorting in section 7.

Next, we define the XPath expression that we claim to sort a given sequence.

DEFINITION 6. *Let $\langle S, \prec_S \rangle$ be a sequence, and $R \subseteq S \times S$ be a partial order on S. With expression $Sort_R(\langle S, \prec_S \rangle)$ we denote the following XPath expression.*

```
if (empty(S))
then ()
else
  for $i in (0 to count(S) - 1)
  return
      for $x in ⟨S, ≺S⟩
      return
          if ($i = CountLessThanR($x, S))
          then $x
          else ()
```

Finally, we prove that the just defined XPath expression *Sort* really sorts a given sequence.

THEOREM 1. *Let $\langle S, \prec_S \rangle$ be a sequence, and $R \subseteq S \times S$ be a partial order on S. Let $\langle S', \prec_{S'} \rangle = Sort_R(\langle S, \prec_S \rangle)$ be a sequence. Then $\langle S', \prec_{S'} \rangle$ is a sequence $\langle S, \prec_S \rangle$ sorted with R.*

Proof. If S is empty then the result of the above expression is an empty sequence, as guaranteed by the outermost if expression. In such a case, the resulting sequence is trivially sorted. In the following, we assume that S is non-empty.

First, we prove that $S' = S$, which is the first condition of the Definition 5 of a sorted sequence. Lemma 3 provides bounds to $CountLessThan_R$ expression, which is $0 \leq CLT_R(x, S) \leq |S| - 1$. Since the outer loop of $Sort_R$ iterates over all values within these bounds, then for each $x \in S$ there exists such i that $i = CLT_R(x, S)$. For such i, item x is returned, thus $S \subseteq S'$. Since CLT_R is a function, there exists exactly one such i for each $x \in S$, therefore $S' = S$.

Second, we prove the condition ii) that requires that $x <_R y$ implies $x \prec_{S'} y$. Let's suppose that $x <_R y$ holds. Then clearly $CLT_R(x, S) < CLT_R(y, S)$, which is obvious from Definition 3 of CLT_R. Since the outer loop of $Sort_R$ iterates over i in a growing manner, x is returned sooner than y, which defines the order of x, and y in the resulting sequence. Thus, we have $x \prec_{S'} y$.

Similarly, we prove the last condition of the definition of a sorted sequence that $x =_R y$ implies ($x \prec_S y$ implies $x \prec_{S'} y$). Suppose that $x =_R y$ and $x \prec_S y$, which means that x and y are equal in terms of the sorting relation R and that x precedes y in the initial sequence. From Lemma 4, we have $CLT_R(x, S) = CLT_R(y, S)$, so x and y are returned in the same iteration of the outer loop of $Sort_R$. From the semantics of the for-return expression comes the following. Since the inner loop iterates over S in the order defined with $\prec_S$, `$x` is bound to x prior to y. Therefore x is returned sooner than y. Thus, we have $x \prec_{S'} y$. □

The proof of this theorem is provided in full detail in [11].

7. FIXING XQUERY SORTING SEMANTICS

In this section, we briefly inspect sorting semantics of XQuery, explaining inconsistency of its definition in W3C's XQuery formal semantics [9]. We explain that consistency can be gained again if we remove the order by clause from XQuery Core, which does not affect the expressive power of the language. This is a consequence of the ability of XPath 2.0 to sort sequences.

Since XQuery allows a lot of syntactic sugaring, a subset called XQuery Core has been extracted from it, on which formal semantics has been defined. We can transform an XQuery query to an XQuery Core query through a process called *normalization*. The Core language is supposed to be equally expressive as the full language, but one can easily find out that it is not. It is sorting, that one cannot express in XQuery Core.

While the grammar of XQuery Core defines a nonterminal for an order by clause, there is no grammar production that refers to it. Further, the normalization of an order by clause of an XQuery query is also omitted in the specification, mentioning only that a data type for a tuple would have to be introduced to define the formal semantics of sorting and since a tuple is not in a data model the specification does not define the formal semantics of sorting at all.

In [11], we show that an order by clause has not to appear in XQuery Core, since i) the semantics of XQuery sorting is based on partial order, ii) we use $Sort_R$ to sort according to that partial order, and iii) tuples can be simulated with other constructs of the language.

Here, we only give an example FLWOR query and its transformation to an equal query without an order by clause. For technical details concerning a general FLWOR query see [11]. Prior to transforming an example query, we put some light on a problem with an order by clause in XQuery Core.

We use the following query in our demonstration.

```
for $a in //author,
    $b in //artist
order by $a/last,
         $b/last,
         $a/first
return <pair>{ $a, $b }</pair>
```

It returns pairs consisting of an author and an artist sorted by author's last name, artist's last name, and author's first name. The sorting condition is somewhat artificial, but we want it that way.

7.1 XQuery Core Fails

How this query should be normalized? A reader familiar with XQuery Core can clearly see that it is normalized to two nested for expressions. But where to put the order by clause? We cannot put it as a whole to the outer for expression that declares variable $a, since the order by clause refers to variable $b that is declared in the inner for expression, which is out of scope. OK, let's try then the inner for expression. Now, both $a and $b are declared, but $a is bound to an author element, so we cannot sort according to author's values. The only chance is to split the order by clause as in the following listing.

```
for $a in //author
order by $a/last, $a/first
return
    for $b in //artist
    order by $b/last
    return
        <pair>{ $a, $b }</pair>
```

We can see that splitting and order by clause can end up in a query that is not equal to the initial one, as for the same author's last name, it sorts first according to author's first name and then according to artist last name, which is not what we desired. This is the reason, why an order by clause should *not* be included in XQuery Core.

7.2 Transformation

The transformation of an FLWOR query to an equal query without an order by clause is rather simple. It is done in three steps: i) we rewrite the initial query not to contain an order by clause, but to contain order values, ii) we define a less-than relation R according to an order by clause from the initial query to be used with our $Sort_R$ expression, and iii) we extract the result.

First, we rewrite the initial query to return a sequence of tuples, where a tuple is an element constructed for each variable binding. It contains a value of every sorting expression from an order by clause and a result value.

```
let $stream := <stream>{
    for $a in //author
    return
        for $b in //artist
        return
            <tuple>
                <ordval>{ $a/last }</ordval>
                <ordval>{ $b/last }</ordval>
                <ordval>{ $a/first }</ordval>
                <result>{
                    <pair>{ $a, $b }</pair>
                }</result>
            </tuple>
    }</stream>
```

Let's make a little technical note about this expression. We create tuple elements as children of a new stream element, as it is the only way we can preserve the information about the order in which $a and $b are bound. If the parent stream element was not created, then the order of tuple elements in the resulting sequence would be implementation dependent.

Second, we transform the order by clause to a less-than relation R, which can be expressed with an XQuery expression as follows.

```
x/ordval[1] lt y/ordval[1] or (
x/ordval[1] eq y/ordval[1] and (
    x/ordval[2] lt y/ordval[2] or (
    x/ordval[2] eq y/ordval[2] and
        x/ordval[3] lt y/ordval[3]
)))
```

Notice, that x and y refer to tuple elements in the example above. The expression is true if the first order value (author's last name) of tuple element x is less than the first order value of tuple element y, or if these values are equal, then it is true if the second order value (artist's last name) of x is less than the second order value of y, and if even these are equal, then third order values are compared (author's first name).

Also notice, that the characteristic function of a lexicographic order defined with an order by clause in a FLWOR expression, can be always expressed with an XQuery expression likewise.

Now, we can sort the sequence of tuples with our $Sort_R$ expression.

```
let $sorted := Sort_R(for $t in $stream/tuple return $t)
```

The $sorted variable now contains a stream of tuple elements sorted according to the same condition as expressed by the order by clause of the initial query.

Finally, we simply pull the results out of this sorted sequence with the following expression.

```
for $t in $sorted
return $t/result/*
```

To sum up, we transformed the initial query to an equal one that has no order by clause.

8. CONCLUSIONS

We demonstrated that XPath 2.0 is a language with great expressive power, which is particularly a come out of introducing the for-return expression and a sequence constructor of atomic integer values. We proved that the power is good enough to sort sequences according to a given partial order, which is a natural means of sorting.

There are two main consequences of sorting capability of XPath.

First, though sorting is expressible in XPath, it is by no means efficient: complexity of $O(n^3)$ in number of items in a sequence being sorted is not satisfactory at all. Therefore a question arises, whether XPath should be extended to express sorting explicitly, which would allow XPath processors use an efficient sorting algorithm.

Second, authors of XQuery formal semantics thought that sorting semantics is not expressible within a data model, which causes an inconsistency in the specification. In this article, we outlined how sorting in XPath can be used to express sorting semantics of XQuery. We referred a more interested reader to a full detail description in [11].

9. ACKNOWLEDGEMENTS

This research was supported in part by GACR grant 201/03/0912. The author's participation at the workshop was partially supported by the "Trust Fund for young DB researchers in the Czech Republic".

10. REFERENCES

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[2] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simeon. XML path language (XPath) 2.0, 2005. W3C Working Draft. `http://www.w3.org/TR/xpath20/`.

[3] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML query language, 2005. W3C Working Draft. `http://www.w3.org/TR/xquery/`.

[4] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible markup language (XML) 1.1, 2004. W3C Recommendation. `http://www.w3.org/TR/xml11/`.

[5] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An xml query language for heterogeneous data sources. In D. Suciu and G. Vossen, editors, *WebDB (Selected Papers)*, volume 1997 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2000.

[6] J. Clark. XSL transformations (XSLT) version 1.0, 1999. W3C Recommendation. `http://www.w3.org/TR/xslt/`.

[7] J. Clark and S. DeRose. XML path language (XPath) version 1.0, 1999. W3C Recommendation. `http://www.w3.org/TR/xpath20/`.

[8] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. XML-QL: A query language for XML. In *WWW The Query Language Workshop (QL)*, 1998.

[9] D. Draper, P. Fankhauser, M. Fernandez, A. Malhotra, Corporation, K. Rose, M. Rys, J. Simeon, and P. Wadler. XQuery 1.0 and XPath 2.0 formal semantics, 2005. W3C Working Draft. `http://www.w3.org/TR/xquery-semantics/`.

[10] M. Fernandez, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 data model, 2005. W3C Working Draft. `http://www.w3.org/TR/xpath-datamodel/`.

[11] P. Hloušek. *XPath, XSLT, and XQuery: Formal Approach*. PhD thesis, Charels University, Prague, 2005. PhD thesis in progress. `http://kocour.ms.mff.cuni.cz/~hlousek/papers/XSXQcomp.pdf`.

[12] A. Malhotra, J. Melton, and N. Walsh. XQuery 1.0 and XPath 2.0 functions and operators, 2005. W3C Working Draft. `http://www.w3.org/TR/xpath-functions/`.