Compiling XPath for Streaming Access Policy

Pierre Genevès*
INRIA Rhône-Alpes
pierre.geneves@inria.fr

Kristoffer Rose
IBM T. J. Watson Research Center
krisrose@us.ibm.com

ABSTRACT

We show how the full XPath language can be compiled into a minimal subset suited for stream-based evaluation. Specifically, we show how XPath normalization into a core language as proposed in the current W3C "Last Call" draft of the XPath/XQuery Formal Semantics can be extended such that both the context state and reverse axes can be eliminated from the core XPath (and potentially XQuery) language. This allows execution of (almost) full XPath on any of the emerging streaming subsets.

Categories and Subject Descriptors: D.3.4 Processors

General Terms: Standardization, Languages, Theory.

Keywords: XPath, streaming, compilation, static rewriting.

1. INTRODUCTION

XPath [1] is emerging as the dominant notation for describing *selection* of nodes in XML data as well as for performing (basic) computations over the values stored in the nodes. The idea of XPath is to navigate XML data in "steps" that each move the "focus" from one node to another. For instance, the XPath expression

```
/descendant::employee/ancestor::manager[1]
```

enumerates all `employee` elements and then collects for each the closest `manager` ancestor element. The language for specifying steps is very rich in what kind of node associations one can use to navigate between them in order to make it as easy as possible to reach any focus of interest from any other.

When XML is stored or transmitted then the system architecture often imposes limitations on what kinds of navigation are efficient for the XML data. Since XPath allows any conceivable access policy then current mainstream XPath implementations such as Xalan implement XPath by copying the entire XML data contents into a linked memory structure such as the Document Object Model. The full XPath language can then easily be supported on top of this in-memory structure by implementing navigation as described in XPath denotational semantics [5]. This approach however suffers from several major drawbacks. It implies heavy memory requirements since the XML data must first fit into memory in order for a query to be processed, and second the entire XML content is loaded whatever the actual query is. Eventually, complexity and efficiency issues arise because the XML tree can be traversed multiple times in different directions.

When storing XML data in a single XML document, then only so-called "streaming" access, where the nodes are visited only once in depth-first tree traversal order, is truly efficient. As such, this access policy has attracted much attention. Several attempts have studied various reduced XPath fragments. Certainly one of the most advanced work is the XSQ streaming XPath engine [4] which supports a significant fragment of XPath, except two hard features that are generally left out of the considered XPath fragments in other studies, although often used in practice. These features are XPath context references (`position()` and `last()` pseudo functions) and reverses axes.

Our approach aims at supporting full XPath on top of a streaming XML infrastructure. In this work we propose a translation of the full XPath language into a minimal subset based on the core expression language of the XPath/XQuery formal semantics [2] . Specifically we show how an XPath expression containing context-state references and/or reverse axes can be compiled into an equivalent one with only forward axes and without context sensitive references. This enables the use of full XPath 1.0 on any streaming implementation. We propose the following staged approach as our translation:

Normalization. Transform the XPath expression into an equivalent expression in the minimal but fully expressive core language specified in the XPath/XQuery formal semantics[2]. This makes the semantics much more explicit by expressing the individual path steps.

Eliminate context position. Rewrite the core XPath expressions to eliminate the implicitly updated context state. References to the context position (and size) are replaced by an expression computing the context position from the context node.

Eliminate reverse axes. Refactor the expression in order to facilitate streaming. Steps involving a reverse axis are converted to steps using the corresponding forward axis.

The full core language and the precise normalization rules are given in the XPath/XQuery formal semantics [2]. Leveraging our translation on the XPath 2 normalization translation makes our job much easier and more transparent. The following two sections explain the transformations: "statelessness" in Section 2 and "forward-only" in Section 3. Figure 4 presents the minimal subset generated by the transformations applied to full XPath 1.0. Finally we conclude in Section 4.

*Work done while visiting IBM T. J. Watson Research Center.

DocEng'05, November 2–4, 2005, Bristol, United Kingdom.

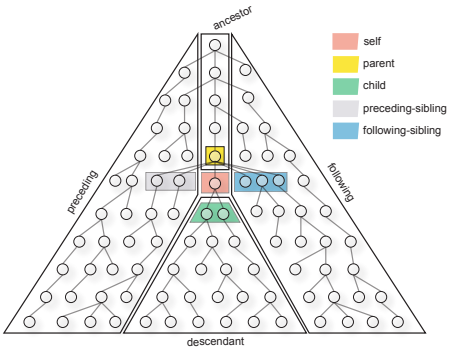

Figure 1: Axis partition for context node.

$S[\![\cdot]\!] : Expr \to (Var \to Expr) \to Expr$

$$S[\![\texttt{let \$}Var_{\text{seq}} \texttt{ := }ForwardAxis\texttt{::}NodeTest\texttt{ return }Expr]\!]\rho = \texttt{let \$}Var_{\text{seq}} \texttt{ := }ForwardAxis\texttt{::}NodeTest\texttt{ return } S[\![Expr]\!](\rho[Var_{\text{seq}} \mapsto ForwardAxis\texttt{::}NodeTest])$$

$$S[\![\texttt{let \$}Var_{\text{seq}} \texttt{ := ddo(}ReverseAxis\texttt{::}NodeTest\texttt{) return }Expr]\!]\rho = \texttt{let \$}Var_{\text{seq}} \texttt{ := }ReverseAxis\texttt{::}NodeTest\texttt{ return } S[\![Expr]\!](\rho[Var_{\text{seq}} \mapsto ReverseAxis\texttt{::}NodeTest])$$

$$S[\![\texttt{let \$}Var_{\text{seq}} \texttt{ := ddo(}Expr_1\texttt{) return }Expr_2]\!]\rho = \texttt{let \$}Var_{\text{seq}} \texttt{ := ddo(}S[\![Expr_1]\!]\rho\texttt{) return }S[\![Expr_2]\!](\rho[Var_{\text{seq}} \mapsto S[\![Expr_1]\!]\rho])$$

$$S[\![\texttt{for \$}Var_{\text{dot}} \texttt{ at \$}Var_{\text{pos}} \texttt{ in \$}Var_{\text{seq}} \texttt{ return }Expr]\!]\rho = \texttt{for \$}Var_{\text{dot}} \texttt{ in \$}Var_{\text{seq}} \texttt{ return let \$}Var_{\text{pos}} \texttt{ := }Expr_{\text{pos}} \texttt{ return } S[\![Expr]\!]\rho$$

where $Expr_{\text{pos}}$ is given by the following table:

$\rho(Var_{\text{seq}})$	$Expr_{\text{pos}}$
`self::`*NodeTest*	1
`child::`*NodeTest*	`count(preceding-sibling::`*NodeTest*`)` + 1
`parent::`*NodeTest*	1
`ancestor::`*NodeTest*	`count(ancestor-or-self::`*NodeTest*`)`
`ancestor-or-self::`*NodeTest*	`count(ancestor-or-self::`*NodeTest*`)`
other *Expr*	`let $`Var_{d} `:= $`Var_{dot} `return count(for $`Var_{dot} `in $`Var_{seq} `return if node-before($`Var_{dot}`,$`Var_{d}`) then $`Var_{dot} `else ()) + 1` where Var_{d} is a fresh variable

Figure 2: Transformation to state-less form.

$F \quad [\![\cdot]\!] : Expr \to Expr$

$F \quad [\![ReverseAxis\texttt{::}NodeTest]\!] =$

```
let $Var_seq := [[/descendant-or-self::NodeTest]]_Expr return
    let $Var_d := $Var_dot return
      for $Var_dot in $Var_seq return
        if exists(intersect($Var_d, ForwardAxis::node()))
          then $Var_dot
            else ()
```

where $[\![]\!]_{\text{Expr}}$ denotes the normalization described in XPath formal semantics[2]; Var_{d} should be chosen fresh; and the *ForwardAxis* is determined from the *ReverseAxis* by this table:

ReverseAxis	*ForwardAxis*
`parent`	`child`
`ancestor`	`descendant`
`ancestor-or-self`	`descendant-or-self`
`preceding-sibling`	`following-sibling`
`preceding`	`following`

Figure 3: Converting reverse steps to forward steps.

Path	p	::=	`let $seq :=`p_1 `return`p_2 | `let $`*en* `:=`*e* `return`p | `let $`*dn* `:= $dot return`p | `for $dot in $seq return`p | `if`b `then $dot else ()` \| `union(`p_1`,`p_2`)` \| `distinct-doc-order(`p`)` | a`::node()` | `self::`*nt* | `root(self::node())` | `subsequence($seq,$`*en*`,1)` | `subsequence($seq,`*e*`,1)` | `()`
Axis	a	::=	`self` | `attribute` | `namespace` | *fa*
ForwardAxis	*fa*	::=	`child` | `descendant` | `descendant-or-self` | `following` | `following-sibling`
NodeTest	*nt*	::=	`*` | *nn* | `node()` | `text()`
Boolean	b	::=	b_1 `or` b_2 | b_1 `and` b_2 | `not(`b`)` | `node-before($dot,$`*dn*`)` | `exists(intersect($`*dn*`,`*fa*`::node()))` \| `exists(`p`)` | *pc*
PathComparison	*pc*	::=	`some $`*dn* `in`p `satisfies`*pc* | `eq(`e_1`,`e_2`)` | `eq(`s_1`,`s_2`)`
Expr	e	::=	`plus(`e_1`,`e_2`)` | `minus(`e_1`,`e_2`)` | `count(`p`)` | `count($seq)` \| `$`*en* | `number(`s`)` | i

where s denotes a string, i an integer, and *nn*, *en*, and *dn* are disjoint identifiers.

Figure 4: XPath subset generated by the transformations.

2. ELIMINATING CONTEXT POSITION

Normalized expressions coming out of the first stage are basically nested "for" constructs where context position occurs implicitly. Specifically, the

`for $dot at $pos in $seq return` *Expr*

construct [2] iterates over the sequence `$seq` (which must be given as a variable) and concatenates all the results of computing the expression, *Expr*, with `$dot` bound to each of the members of the sequence value, in order, and `$pos` bound to the index number of each member in the sequence (corresponding to the value of the `position()` pseudo-function in XPath).

The elimination of the context position is important because it frees implementations from following the notion of "iteration over a sequence" operationally since index numbers are just values like any other – in a sense the expressions that come out of the context position elimination stage are as data access policy independent as possible.

To eliminate state we must modify the XPath expression into another expression without any use of the `at` binder in `for` constructions. This is achieved by:

- Keeping track for every node sequence variable what the defining step is.
- For every occurrence of `at` replace it with an explicit `let` binding to a computation of the index.

The basic idea is that the position can be calculated from expressions relative to the current node and the node that generated the context sequence. This is because these two nodes define a clean partitioning of the complete collection of nodes as illustrated on figure 1. The translation *S* is formally specified as a "derivor" shown on figure 2 and written $S[\![Expr]\!]\rho$, where:

- The *Expr* parameter is the one that is rewritten (and since it is source language syntax we surround it with the special "syntax" braces $[\![]\!]$).
- The additional parameter ρ maps all node sequence variables that are in scope to the axis and node-test used.

The environment ρ supports two operations:

- $\rho[Var \mapsto Axis::NodeTest]$ returns a new environment which is like the ρ environment except it includes a description that the *Var* variable is a node sequence constructed using the $Axis::NodeTest$ path step.
- $\rho(Var)$ denotes the most recent pair $Axis::NodeTest$ added to the ρ environment for `$`*Var*.

The translations show how the position index can be recomputed for every node by counting the nodes in the context sequence that occur before the context node except in the (few but common) cases where the XPath axis symmetries provide a more direct way to compute the count, as shown on table of figure 2.

Note that the derivor is only specified formally for the interesting cases; for all other expression forms it is just distributed over the subexpressions, *e.g.*,

$$S[\![Expr_1 + Expr_2]\!]\rho = S[\![Expr_1]\!]\rho \ + \ S[\![Expr_2]\!]\rho$$

Finally, we observe the inherent limitation in this approach: the global context state is not eliminated. Thus instances of `position()` and `last()` used at the top level of XPath queries should not be translated but merely return the global context's position and size, respectively.

3. ELIMINATING REVERSE AXES

The third and final transformation is about making sure that only forward axes are used. The elimination of reverse axes also relies on the symmetries illustrated in Figure 1. In fact it is closely based on equivalences like those of the "looking forward" analysis [3] extended to handle variable binding. The nodes in the sequence constructed by the reverse axis are instead obtained by *searching* for ways to reach the context node from any node, using the symmetric forward axis. The search succeeds for a reverse axis node if the intersection of the converse forward axis finds the context node from the candidate node. Again we have only specified the interesting case on figure 3: the translation of an actual reverse step.

4. CONCLUSION

We have shown how a XPath 1.0 node selection (path expression) can be translated into a form with no explicit use of non-top-level context position (or size) and using only forward axes. The subset of the core language generated by the transformations is shown on figure 4.

Our approach has however certain limitations. First, the context position elimination transformation relies on the property of XPath 1.0 that all sequences that are indexed (with `position()`) must be defined by a single XPath "step" expression with an explicit axis and node test. Second, we rely on the efficiency of a few primitive operations that are not in XPath 1.0, namely document order comparison (`node-before`) and intersection (`intersect`). These are highly efficient in streaming implementations but may be costly in other contexts. Thus while the approach is certainly viable for a streaming XPath 1.0 implementation it is not clear how useful the framework could be for other combinations of primitives. We would like to investigate in particular how much of the full core XPath 2.0/XQuery [2] can be implemented in this way.

In future work we will relate the translation to the formal semantics of XPath (and XQuery) [2] as well as the classical semantics of Wadler [5]: in terms of a formal semantics we would like to prove that $F[\![S[\![Expr]\!]\,\emptyset]\!]$ evaluates to the same values as *Expr* in any context, that is, in any dynamic environment where an XPath expression evaluates to a result value, the same result value can be obtained from the transformed XPath.

We also plan to report runtimes of using the performance results in connection with streaming by using the various streaming subsets as backends for the transformation.

5. REFERENCES

[1] J. Clark and S. DeRose. XML path language (XPath) version 1.0, W3C recommendation, November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[2] 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, W3C working draft, June 2005. http://www.w3.org/TR/xquery-semantics/.

[3] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. In *Proc. of the EDBT Workshop on XML Data Management (XMLDM)*, volume 2490 of *LNCS*, pages 109–127. Springer, 2002.

[4] F. Peng and S. S. Chawathe. Xpath queries on streaming data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 431–442, New York, NY, USA, 2003. ACM Press.

[5] P. Wadler. Two semantics for XPath, January 2000. http://homepages.inf.ed.ac.uk/wadler/papers/xpath-semantics/xpath-semantics.pdf.