Embedding XQuery in Toy *

Jesús Almendros-Jiménez [§], Rafael Caballero[†], Yolanda García-Ruiz[†] and Fernando Sáenz-Pérez[‡]

Technical Report SIC-04-11

[§]Dpto. Lenguajes y Computación, Universidad de Almería, Spain
[†]Dpto. de Sistemas Informáticos y Computación, UCM, Spain
[‡]Dpto. de Ingeniería del Software e Inteligencia Artificial, UCM, Spain
jalmen@ual.es, {rafa,fernan}@sip.ucm.es , ygarciar@fdi.ucm.es

April 2011
(Revised February 10, 2014)

* Work partially supported by the Spanish projects STAMP TIN2008-06622-C03-01, Prometidos-CM S2009TIC-1465 and GPD UCM-BSCH-GR58/08-910502.

Abstract. This report addresses the problem of integrating a fragment of XQuery, a language for querying XML documents, into the functional-logic language $\mathcal{TOY}$. The queries are evaluated by an interpreter, and the declarative nature of the proposal allows us to prove correctness and completeness with respect to the semantics of the subset of XQuery considered. The different fragments of XML that can be produced by XQuery expressions are obtained using the non-deterministic features of functional-logic languages. As an application of this proposal we show how the typical *generate and test* techniques of logic languages can be used for generating test-cases for XQuery expressions.

1 Introduction

XQuery has been defined as a query language for finding and extracting information from XML [19] documents. Originally designed to meet the challenges of large-scale electronic publishing, XML also plays an important role in the exchange of a wide variety of data on the Web and elsewhere. For this reason many modern languages include libraries or encodings of XQuery, including logic programming [1] and functional programming [9]. In this report we consider the introduction of a simple subset of XQuery [5,21] into the functional-logic language $\mathcal{TOY}$ [14].

One of the key aspects of declarative languages is the emphasis they pose on the logic semantics underpinning declarative computations. This is important for reasoning about computations, proving properties of the programs or applying declarative techniques such as abstract interpretation [7,8], partial evaluation [13] or algorithmic debugging [18]. There are two different declarative alternatives that can be chosen for incorporating XML into a (declarative) language:

1. Use a domain-specific language and take advantage of the specific features of the host language. This is the approach taken in [11], which presents a rule-based language for processing semistructured data that is implemented and embedded in the functional logic language Curry, and also in [17] for the case of logic programming.
2. Consider an existing query language such as XQuery, and embed a fragment of the language in the host language, in this case $\mathcal{TOY}$. This is the approach considered in this report.

Thus, our goal is to include XQuery using the purely declarative features of the functional-logic language $\mathcal{TOY}$. Moreover, analyzing the functional-logic semantics [15] of the embedding we are able to prove that the semantics of the considered fragment of XQuery has been correctly included in $\mathcal{TOY}$. To the best of our knowledge, it is the first time a fragment of XQuery has been encoded in a functional-logic language. A first step in this direction was proposed in [4], where XPath [6] expressions were introduced in $\mathcal{TOY}$. XPath is a subset of XQuery that allows navigating and returning fragments of documents in a similar way as the path expressions used in the *chdir* command of many operating systems. The contributions of this report with respect to [4] are:

1. The setting has been extended to deal with a simple fragment of XQuery, including *for* statements for traversing XML sequences, *if/where* conditions, and the possibility of returning XML elements as results. Some basic XQuery constructions such as *let* statements are not considered, but we think that the proposal is powerful enough for representing many interesting queries.
2. The soundness of the approach is formally proved, checking that the semantics of the fragment of XQuery included in our setting is correctly represented in $\mathcal{TOY}$.

Next Chapter introduces the fragment of XQuery considered and a suitable operational semantics for evaluating queries. Then the language $\mathcal{TOY}$ and its

$$
\begin{array}{rl}
\mathit{query} ::= & \mathit{query}\ \mathit{query} \mid \mathit{tag} \\
\mid & \mathit{var} \ \mid\ \mathit{var}/\mathit{axis} :: \nu \\
\mid & \textbf{for}\ \mathit{var}\ \textbf{in}\ \mathit{query}\ \textbf{return}\ \mathit{query} \\
\mid & \textbf{if}\ \mathit{cond}\ \textbf{then}\ \mathit{query} \\
\mathit{cond} ::= & \mathit{var} = \mathit{var} \mid \mathit{query} \\
\mathit{tag} ::= & \langle a \rangle\ \langle /a \rangle \mid \langle a \rangle \mathit{var}\ \ldots\ \mathit{var} \langle /a \rangle \mid \langle a \rangle \mathit{tag} \langle /a \rangle \\
\mathit{axis} ::= & \textbf{self} \mid \textbf{child} \mid \textbf{descendant} \mid \textbf{dos}
\end{array}
$$

Fig. 1. Syntax of SXQ, a simplified version of XQ

semantics are presented in Chapter 3. Chapter 4 includes the interpreter that performs the evaluation of simple XQuery expressions in $\mathcal{TOY}$. The theoretical results establishing the soundness of the approach with respect to the operational semantics of Chapter 2 are presented in Section 4.1. Chapter 5 explains the automatic generation of Test Cases for simple XQuery expressions. Finally, Chapter 6 concludes summarizing the results and proposing future work.

2 XQuery and Its Operational Semantics

XQuery allows the user to query several documents, applying join conditions, generating new XML fragments, and many other features [5,21]. The syntax and semantics of the language are quite complex [20], and thus only a small subset of the language is usually considered. The next section introduces the fragment of XQuery considered in this report.

2.1 The subset SXQ

In [3] a declarative subset of XQuery, called XQ, is presented. This subset is a core language for XQuery expressions consisting of *for*, *let* and *where/if* statements. In this report we consider a simplified version of XQ, which we call SXQ and whose syntax can be found in Figure 1. In this grammar, a denotes a label and ν refers to a *label test* which is a label. The differences of SXQ with respect to XQ are:

1. XQ includes the possibility of using variables as tag names using a constructor *lab($x)*.
2. XQ permits enclosing any query Q between tag labels $\langle a \rangle Q \langle /a \rangle$. SXQ admits a start and end tag with nothing between them $\langle a \rangle \langle /a \rangle$ and either variables or other tags inside a tag.
3. XQ allows the empty query ( ) as a valid query. This allows representing expressions of the form $\langle a \rangle \langle /a \rangle$. Although SXQ does not allow empty queries, these expressions are built in SXQ by the *tag* constructor.

Our setting can be easily extended to support the *lab($x)* feature, but we omit this case for the sake of simplicity in this presentation. The second restriction is more severe: although *let*s are not part of XQ, they could be simulated using *for* statements inside tags. In our case, forbidding other queries different from variables inside tag structures imply that our core language cannot represent *let* expressions. This limitation is due to the non-deterministic essence of our embedding, since a *let* expression means collecting all the results of a query instead of producing them separately using non-determinism. In spite of these limitations, the language SXQ is still useful for solving many common queries as the following example shows.

Example 1. Consider an XML file "bib.xml" containing data about books, and another file "reviews.xml" containing reviews for some of these books (see Appendix A). Then, we can list the reviews corresponding to books in "bib.xml" as follows:

```
for $b in doc("bib.xml")/bib/book,
    $r in doc("reviews.xml")/reviews/entry
where $b/title = $r/title
for $booktitle in $r/title,
    $revtext in $r/review
return <rev> $booktitle  $revtext </rev>
```

The variable $\$b$ takes the value of each different book, and $\$r$ represents the different reviews. The *where* condition ensures that only reviews corresponding to the book are considered. Finally, the last two variables are only employed to obtain the book title and the text of the review, the two values that are returned as output of the query by the *return* statement.

It can be argued that the code of this example does not follow the syntax of Figure 1. While this is true, it is very easy to define an algorithm that converts a query formed by *for*, *where* and *return* statements into a SXQ query (as long as it only includes variables inside tags, as stated above). The idea is simply to replace the references to XML documents by new indexed variables $\$x_1, \$x_2, \ldots,$ and convert the *where* into *if*s, following each *for* by a *return*, and decomposing XPath expressions including several steps into several *for* expressions by introducing a new auxiliary variables, each one consisting of a single step.

Example 2. The query of Example 1 using SXQ syntax:

```
for $x3 in $x1/child::bib return
for $x4 in $x3/child::book  return
for $x5 in $x2/child::reviews return
for $x6 in $x5/child::entry  return
for $x7 in $x4/child::title return
for $x8 in $x6/child::title return
if ($x7 = $x8) then
   for $x9 in $x6/child::title return
     for $x10 in $x6/child::review return <rev> $x9  $x10 </rev>
```

Notice that the expressions doc("bib.xml") and doc("reviews.xml") have been substituted by the variables `$x1` and `$x2` respectively. Both variables are free in the query and the value of each one is the XML document contained in the corresponding XML file.

The concept of set of *free* variables of a SXQ query is given by the following inductive definition:

Definition 1. *Let Q be a SXQ query. The set of free variables of Q, denoted by free(Q), is defined as follows:*

- *If $Q \equiv Q_1\ Q_2$, then free$(Q) :=$ free$(Q_1) \cup$ free(Q_2)*
- *If $Q \equiv$ $\mathsf{for}\ \ \$x\ \mathsf{in}\ \ Q_1\ \mathsf{return}\ \ Q_2$, then free$(Q) := ($free$(Q_1) \cup$ free$(Q_2)) \setminus \{\$x\}$*
- *If $Q \equiv$ $\mathsf{if}\ \ \$x_i = \$x_j\ \mathsf{then}\ \ Q_2$, then free$(Q) := \{\$x_i, \$x_j\} \cup$ free(Q_2)*
- *If $Q \equiv$ $\mathsf{if}\ \ Q_1\ \mathsf{then}\ \ Q_2$, then free$(Q) :=$ free$(Q_1) \cup$ free(Q_2)*
- *If $Q \equiv \$x$, then free$(Q) := \{\$x\}$*
- *If $Q \equiv \$x/axis :: \nu$, then free$(Q) := \{\$x\}$*
- *If $Q \equiv \langle a\rangle\langle/a\rangle$, then free$(Q) := \emptyset$*
- *If $Q \equiv \langle a\rangle tag\langle/a\rangle$, then free$(Q) :=$ free(tag)*
- *If $Q \equiv \langle a\rangle\$x_i \ldots \$x_j\langle/a\rangle$, then free$(Q) := \{\$x_i, \ldots, \$x_j\}$*

We assume that XML documents must be accessed initially via indexed variables $\$x_1, \$x_2, \ldots$ belonging to the set *free*(Q).

Next we define a function ρ that takes a SXQ query and an index m (a positive integer) as inputs and returns a new SXQ query equivalent to Q.

Definition 2. *Let Q be a SXQ query and an index m. We define the query $\rho(Q, m)$ from Q recursively as follows:*

1. *If $Q \equiv Q_1\ Q_2$, then $\rho(Q, m) := \rho(Q_1, m)\rho(Q_2, m)$*
2. *If $Q \equiv$ $\mathsf{for}\ \ \$a\ \mathsf{in}\ \ Q_1\ \mathsf{return}\ \ Q_2$, then*

$$\rho(Q, m) := \ \mathsf{for}\ \ \$x_{m+1}\ \mathsf{in}\ \ \rho(Q_1, m)\ \mathsf{return}\ \ \rho(Q_2', m+1)$$

   where Q_2' is the SXQ query obtained from Q_2 by replacing all occurrences of variable $\$a$ by the variable $\$x_{m+1}$.
3. *If $Q \equiv$ $\mathsf{if}\ \ Q_1\ \mathsf{then}\ \ Q_2$, then*

$$\rho(Q, m) := \ \mathsf{if}\ \ \rho(Q_1, m)\ \mathsf{then}\ \ \rho(Q_2, m)$$

4. *If $Q \equiv$ $\mathsf{if}\ \ \$x_i = \$x_j\ \mathsf{then}\ \ Q_1$, then*

$$\rho(Q, m) := \ \mathsf{if}\ \ \$x_i = \$x_j\ \mathsf{then}\ \ \rho(Q_1, m)$$

5. *In the rest of the cases, $\rho(Q, m) := Q$.*

Without loss of generality, in the rest of the report we assume that all the variables occurring in Q are renamed following the Definition 2 using index k with k the cardinality of *free*(Q).

Example 3. Consider the XML file “bib.xml” from Example 1. Let Q be the following SXQ query:

```
for $b in for $c in $x1/child::bib return $c/child::book return
for $c in $b/child::title return
for $d in $b/child::price return  <item>  $c  $d </item>
```

Query Q selects the title and price of books in “bib.xml”. The XML document “bib.xml” is accessed via the variable $\$x1$. By Definition 1, $\mathit{free}(Q) = \{\$x1\}$. The following query is obtained from Q by renaming all the variables occurring in Q according to Definition 2 using index $m = 1$:

```
for $x2 in for $x2 in $x1/child::bib return $x2/child::book return
for $x3 in $x2/child::title return
for $x4 in $x2/child::price return  <item>  $x3  $x4 </item>
```

Notice that the two occurrences of variable $\$x2$ correspond to different scopes.

We end this Section with a few definitions that are useful for the rest of the report. Following the syntax of Figure 1, SXQ queries can contain subqueries. In that case given a query Q, we use the notation $Q_{|p}$ for representing the subquery Q' that can be found in Q at position p. More formally, notions like positions and subqueries can be defined by induction on the structure of the query.

Definition 3. *Let Q be a SXQ query:*

1. *The set of positions of the query Q is a set $Pos(Q)$ of strings over the alphabet $\{1, 2\}$, which is inductively defined as follows:*
   - *If $Q \equiv \$x$, $Q \equiv \$x/axis :: \nu$, $Q \equiv \langle a\rangle\langle/a\rangle$ or $Q \equiv \langle a\rangle\$x_i \ldots \$x_j\langle/a\rangle$, then $Pos(Q) := \{\varepsilon\}$, where ε denotes de empty string.*
   - *If $Q \equiv \langle a\rangle tag\langle/a\rangle$, then $Pos(Q) := \{\varepsilon\} \cup \{1 \cdot p \mid p \in Pos(tag)\}$.*
   - *If $Q \equiv Q_1\ Q_2$, $Q \equiv$ $\mathsf{for}\ \ \$x_j\ \mathsf{in}\ \ Q_1\ \mathsf{return}\ \ Q_2$ or $Q \equiv$ $\mathsf{if}\ \ Q_1\ \mathsf{then}\ \ Q_2$, then $Pos(Q) := \{\varepsilon\} \cup \bigcup_{i=1}^{2} \{i \cdot p \mid p \in Pos(Q_i)\}$.*
   - *If $Q \equiv$ $\mathsf{if}\ \ \$x_i = \$x_j\ \mathsf{then}\ \ Q_1$, then $Pos(Q) := \{\varepsilon\} \cup \{1 \cdot p \mid p \in Pos(Q_1)\}$.*

   The prefix order defined as

   $$p \le q \textit{ iff there exists } p' \textit{ such that } p \cdot p' = q$$

   is a partial order on positions.
2. *For $p \in Pos(Q)$, the subquery of Q at position p, denoted by $Q_{|p}$, is defined by induction on the length of p:*

   $$\begin{array}{ll} Q_{|\varepsilon} & := Q, \\ (Q_1\ Q_2)_{|i\cdot q} & = (Q_i)_{|q}, \\ (\mathsf{for}\ \$x_k\ \mathsf{in}\ \ Q_1\ \mathsf{return}\ \ Q_2)_{|i\cdot q} & := (Q_i)_{|q}, \\ (\mathsf{if}\ \ Q_1\ \mathsf{then}\ \ Q_2)_{|i\cdot q} & := (Q_i)_{|q}, \\ (\mathsf{if}\ \ \$x_k = \$x_j\ \mathsf{then}\ \ Q_1)_{|1\cdot q} & := (Q_1)_{|q}, \\ (\ \langle a\rangle tag\langle/a\rangle\ )_{|1\cdot q} & := (tag)_{|q}. \end{array}$$

   Note that, for $p = i \cdot q$, $p \in Pos(Q)$ and queries of the form $\langle a\rangle tag\langle/a\rangle$ and $\mathsf{if}\ \ \$x_k = \$x_j\ \mathsf{then}\ \ Q_1$, implies that $i = 1$.

Example 4. The position $p = 2{\cdot}2{\cdot}2{\cdot}2$ of the query in the Example 2 corresponds to the query:

```
for $x7 in $x4/child::title return
  for $x8 in $x6/child::title return
   if ($x7 = $x8) then
    for $x9 in $x6/child::title return
      for $x10 in $x6/child::review return <rev> $x9  $x10 </rev>
```

while the position $p = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 1$ of the same query corresponds to the query:

```
$x4/child::title
```

The set of positions of a query Q is closed under taking prefixes, i.e. if $p \in Pos(Q)$ then $q \in Pos(Q)$ for all $q \leq p$.

Next definition relates the subquery of Q at position $p \in Pos(Q)$ to the variables introduced by *for* statements in Q at positions $q \in Pos(Q)$ with $q < p$.

Definition 4. *Given a SXQ query Q and a position $p \in Pos(Q)$, the set of variables $V_{for}(Q, p)$ is defined as:*

1. $V_{for}(Q, \varepsilon) := \emptyset$
2. $V_{for}(Q_1\ Q_2\ ,\ i \cdot q) := V_{for}(Q_i, q)$
3. $V_{for}(\ \mathsf{for}\ \ \$x_k\ \mathsf{in}\ \ Q_1\ \mathsf{return}\ \ Q_2\ ,\ 1 \cdot q) := V_{for}(Q_1, q)$
4. $V_{for}(\ \mathsf{for}\ \ \$x_k\ \mathsf{in}\ \ Q_1\ \mathsf{return}\ \ Q_2\ ,\ 2 \cdot q) := \{\$x_k\} \cup V_{for}(Q_2, q)$
5. $V_{for}(\ \mathsf{if}\ \ Q_1\ \mathsf{then}\ \ Q_2\ ,\ i \cdot q) := V_{for}(Q_i, q)$
6. $V_{for}(\ \mathsf{if}\ \ \$x_k = \$x_j\ \mathsf{then}\ \ Q_1\ ,\ 1 \cdot q) := V_{for}(Q_1, q)$
7. $V_{for}(\langle a \rangle tag \langle /a \rangle\ ,\ 1 \cdot q) := \emptyset$

Next, we define the set of *relevant* variables for a query Q at position p, denoted by $Rel(Q, p)$, as the set of variables that can appear free in a query Q at position p. The next lemma introduces a basic property of $V_{for}(Q, p)$.

Lemma 1. *Let Q be a SXQ query, and p, q be strings over the alphabet $\{1, 2\}$. If $p \cdot q \in Pos(Q)$, then $V_{for}(Q, p \cdot q) = V_{for}(Q, p) \cup V_{for}(Q_{|p}, q)$.*

Proof. Notice that if $p \cdot q \in Pos(Q)$, then $p \in Pos(Q)$ and $q \in Pos(Q_{|p})$. The result can be proved by induction on the length of p according to the formal definitions given above.

- For $p = \varepsilon$, we have $Q_{|\varepsilon} = Q$. In addition $p = \varepsilon$ implies $p \cdot q = q$ and $V_{for}(Q, p \cdot q) = V_{for}(Q, q)$. By Definition 4, rule 1, $V_{for}(Q, \varepsilon) = \emptyset$. Then, $V_{for}(Q, p \cdot q) = V_{for}(Q, q) = \emptyset \cup V_{for}(Q, q) = V_{for}(Q, \varepsilon) \cup V_{for}(Q_{|\varepsilon}, q)$, which shows the result.
- Now, assume that $p = i \cdot p'$. Because $i \cdot p' \cdot q \in Pos(Q)$, the query Q is of the form:
  - $Q \equiv Q_1\ Q_2$. In this case, i can be either 1 or 2. Suppose $i = 1$. Then:
    (a) $Q_{|p} = Q_{|1 \cdot p'} = (Q_1)_{|p'}$ (by Definition 3)
    (b) $V_{for}(Q, 1 \cdot p') = V_{for}(Q_1, p')$ (by Definition 4, rule 2)

(c) $V_{for}(Q, p \cdot q) = V_{for}(Q, 1 \cdot p' \cdot q) = V_{for}(Q_1, p' \cdot q)$ (by Definition 4, rule 2)

Applying induction on (c) we obtain

(d) $V_{for}(Q_1, p' \cdot q) = V_{for}(Q_1, p') \cup V_{for}((Q_1)_{|p'}, q)$

and by (a) and (b),

(e) $V_{for}(Q_1, p' \cdot q) = V_{for}(Q, 1 \cdot p') \cup V_{for}(Q_{|1 \cdot p'}, q)$

Then, with (c) and (e), we obtain $V_{for}(Q, p \cdot q) = V_{for}(Q, 1 \cdot p') \cup V_{for}(Q_{|1 \cdot p'}, q) = V_{for}(Q, p) \cup V_{for}(Q_{|p}, q)$ and the result holds.

If $i = 2$, the result can be proved similarly.

- $Q \equiv$ for $\$x_k$ in Q_1 return Q_2. If $i = 1$ the result can be proved similarly to the previous case. Now, suppose $i = 2$. Then:

  (a) $Q_{|p} = Q_{|2 \cdot p'} = (Q_2)_{|p'}$ (by Definition 3)

  (b) $V_{for}(Q, p) = V_{for}(Q, 2 \cdot p') = \{\$x_k\} \cup V_{for}(Q_2, p')$ (by Definition 4, rule 4)

  By Definition 4, rule 4, $V_{for}(Q, p \cdot q) = V_{for}(Q, 2 \cdot p' \cdot q) = \{\$x_k\} \cup V_{for}(Q_2, p' \cdot q)$, and by induction we obtain:

  (c) $V_{for}(Q, p \cdot q) = \{\$x_k\} \cup V_{for}(Q_2, p') \cup V_{for}((Q_2)_{|p'}, q)$

  Then, by (a) and (b), the result:

  $V_{for}(Q, p \cdot q) = V_{for}(Q, p) \cup V_{for}((Q_2)_{|p'}, q) = V_{for}(Q, p) \cup V_{for}(Q_{|p}, q)$ holds.

  The rest of the cases can be proved similarly.

Definition 5. *Given a SXQ query Q and a position $p \in Pos(Q)$, $Rel(Q, p)$ is defined as:*

$$Rel(Q, p) := free(Q) \cup V_{for}(Q, p)$$

Example 5. Let Q be the SXQ query in the Example 2 and $p = 2 \cdot 2$ a position in $Pos(Q)$. The subquery $Q_{|p}$ corresponds to the expression:

```
for $x5 in $x2/child::reviews return
 for $x6 in $x5/child::entry  return
  for $x7 in $x4/child::title return
   for $x8 in $x6/child::title return
    if ($x7 = $x8) then
     for $x9 in $x6/child::title return
       for $x10 in $x6/child::review return <rev> $x9  $x10 </rev>
```

The set of variables $free(Q)$ contains the only two variables $\{\$x_1, \$x_2\}$ and following Definition 1, $free(Q_{|p}) = \{\$x_2, \$x_4\}$. Notice variable $\$x_1 \in free(Q)$ but $\$x_1 \notin free(Q_{|p})$. By Definition 4, $V_{for}(Q, p) = \{\$x_3, \$x_4\}$ and following Definition 5, $Rel(Q, p) = \{\$x_1, \$x_2, \$x_3, \$x_4\}$.

Note that, the set $Rel(Q, p)$ collects all the free variables occurring in the query Q and all the variables introduced by *for* statements in positions q with $q < p$. Thus, the set $Rel(Q, p)$ contains all the variables that could appear free in the subquery $Q_{|p}$. Next Lemma formalizes this idea.

Lemma 2. *Let Q be a SXQ query and $p \in Pos(Q)$ be a position of the query Q. If $\$x \in free(Q_{|p})$, then $\$x \in Rel(Q, p)$.*

Proof. The result can be proved by induction on the length of p.

- For $p = \varepsilon$, we have $Q_{|\varepsilon} == Q$. By Definition 5, $Rel(Q, \varepsilon) := free(Q) \cup V_{for}(Q, \varepsilon)$, and by Definition 4, item 1, $Rel(Q, \varepsilon) := free(Q) \cup \emptyset = free(Q_{|\varepsilon})$ which shows the result.
- Now, assume that $p = i \cdot q$. Because $i \cdot q \in Pos(Q)$, the query Q is of the form:
  - $Q \equiv Q_1\ Q_2$. In this case, i can be either 1 or 2. Suppose $i = 1$. Then:
    (a) By Definition 5, $Rel(Q, 1{\cdot}q) = free(Q) \cup V_{for}(Q, 1{\cdot}q)$ and by Definition 4, item 2, $Rel(Q, 1 \cdot q) = free(Q) \cup V_{for}(Q_1, q)$.
    (b) $free(Q_{|1\cdot q}) = free(Q_{1_{|q}})$. If $x \in free(Q_{|1\cdot q})$, then $x \in free(Q_{1_{|q}})$. By induction we obtain $x \in Rel(Q_1, q)$. By Definition 5, $Rel(Q_1, q) = free(Q_1) \cup V_{for}(Q_1, q)$. Now, $x \in Rel(Q_1, q)$ implies either $x \in free(Q_1)$ or $x \in V_{for}(Q_1, q)$.
      - If $x \in free(Q_1)$, then $x \in free(Q_{|1})$. Applying induction we obtain $x \in Rel(Q, 1)$. By Definition 5, $Rel(Q, 1) = free(Q) \cup V_{for}(Q, 1) = free(Q) \cup V_{for}(Q_1, \varepsilon) = free(Q) \cup \emptyset$ (by Definition 4). Then, $x \in free(Q)$ and by (a) $x \in Rel(Q, 1 \cdot q)$ which shows the result.
      - If $x \in V_{for}(Q_1, q)$, then, by (a), $x \in Rel(Q, 1 \cdot q)$ and the result holds.

    If $i = 2$, the result can be proved similarly.

  - $Q \equiv$ for $\$x_k$ in Q_1 return Q_2. If $i = 1$ the result can be proved similarly to the previous case. Now, suppose $i = 2$. Then:
    (a) $Q_{|p} = Q_{|2\cdot q} = (Q_2)_{|q}$ (by Definition 3)
    (b) By Definition 5, $Rel(Q, 2{\cdot}q) = free(Q) \cup V_{for}(Q, 2{\cdot}q)$ and by Definition 4, item 4, $Rel(Q, 2 \cdot q) = free(Q) \cup \{\$x_i\} \cup V_{for}(Q_2, q)$.
    (c) $free(Q_{|2\cdot q}) = free(Q_{2_{|q}})$. If $x \in free(Q_{|2\cdot q})$, then $x \in free(Q_{2_{|q}})$. By induction we obtain $x \in Rel(Q_2, q)$. By Definition 5, $Rel(Q_2, q) = free(Q_2) \cup V_{for}(Q_2, q)$. Now, $x \in Rel(Q_2, q)$ implies either $x \in free(Q_2)$ or $x \in V_{for}(Q_2, q)$.
      - If $x \in free(Q_2)$, then $x \in free(Q_{|2})$. Applying induction we obtain $x \in Rel(Q, 2)$. By Definition 5, $Rel(Q, 2) = free(Q) \cup V_{for}(Q, 2) = free(Q) \cup \{\$x_i\} \cup V_{for}(Q_2, \varepsilon) = free(Q) \cup \{\$x_i\} \cup \emptyset$ (by Definition 4). Then, $\$x \in Rel(()Q, 2)$ implies either $\$x \in free(Q)$ or $\$x = \$x_i$ and by (b), the result holds.
      - If $x \in V_{for}(Q_2, q)$, then, by (b), $x \in Rel(Q, 2 \cdot q)$ and the result holds.

    The rest of the cases can be proved similarly.

Next lemma presents some properties of the set of relevant variables for a query Q at position p.

Lemma 3. *Let Q' be a SXQ query and $p \cdot i \in Pos(Q')$ be a position of the query Q' such that $Q'_{|p} = Q$ and $i \in \{1, 2\}$. Then,*

- *If $Q \equiv$ for $\$x$ in Q_1 return Q_2, then:*
  - $Rel(Q', p \cdot 1) = Rel(Q', p)$
  - $Rel(Q', p \cdot 2) = Rel(Q', p) \cup \{\$x\}$
- *For the rest of the cases, $Rel(Q', p \cdot i) = Rel(Q', p)$*

Proof. We distinguish cases depending of the form of the query Q.

- $Q \equiv Q_1\ Q_2$.
  - If $i = 1$, $Q'_{|p \cdot 1} \equiv Q_{|1} \equiv Q_1$ is an SXQ query. Note that, $p \cdot 1 \in Pos(Q')$. Then,

$$\begin{array}{ll} Rel(Q', p \cdot 1) & = \text{(by Definition 5)} \\ free(Q') \cup V_{for}(Q', p \cdot 1) & = \text{(by Lemma 1)} \\ free(Q') \cup V_{for}(Q', p) \cup V_{for}(Q'_{|p}, 1) & = \text{(by Definition 4, rule 2)} \\ free(Q') \cup V_{for}(Q', p) \cup V_{for}(Q'_{|p \cdot 1}, \varepsilon) & = \text{(by Definition 4, rule 1)} \\ free(Q') \cup V_{for}(Q', p) \cup \emptyset & = \text{(by Definition 5)} \\ Rel(Q', p) & \end{array}$$

  - If $i = 2$, the result $Rel(Q', p \cdot 1) = Rel(Q', p)$ can be proved similarly to the previous case.
- $Q \equiv$ for $\$x$ in Q_1 return Q_2. This query introduces a new variable by means of a *for* statement.
  - In the case of $i = 1$, $Q'_{|p \cdot 1} \equiv Q_1$. Then,

$$\begin{array}{ll} Rel(Q', p \cdot 1) & = \text{(by Definition 5)} \\ free(Q') \cup V_{for}(Q', p \cdot 1) & = \text{(by Lemma 1)} \\ free(Q') \cup V_{for}(Q', p) \cup V_{for}(Q'_{|p}, 1) & = \text{(by Definition 4, rule 3)} \\ free(Q') \cup V_{for}(Q', p) \cup V_{for}(Q'_{|p \cdot 1}, \varepsilon) & = \text{(by Definition 4, rule 1)} \\ free(Q') \cup V_{for}(Q', p) \cup \emptyset & = \text{(by Definition 5)} \\ Rel(Q', p) & \end{array}$$

    which shows the result.
  - In the case of $i = 2$, $Q'_{|p \cdot 2} \equiv Q_2$. Then,

$$\begin{array}{ll} Rel(Q', p \cdot 2) & = \text{(by Definition 5)} \\ free(Q') \cup V_{for}(Q', p \cdot 2) & = \text{(by Lemma 1)} \\ free(Q') \cup V_{for}(Q', p) \cup V_{for}(Q'_{|p}, 2) & = \text{(by Definition 4, rule 4)} \\ free(Q') \cup V_{for}(Q', p) \cup \{\$x\} \cup V_{for}(Q'_{|p \cdot 2}, \varepsilon) & = \text{(by Definition 4, rule 1)} \\ free(Q') \cup V_{for}(Q', p) \cup \{\$x\} \cup \emptyset & = \text{(by Definition 5)} \\ Rel(Q', p) \cup \{\$x\} & \end{array}$$

    which shows the result.

The rest of the cases are analogous to the previous cases.

2.2 XQ Operational Semantics

The semantics of XQ can be found in [3]. We will use XML documents represented as *data trees*. A *data forest* is a sequence of data trees and an *indexed forest* is a pair consisting of a data forest and a sequence of nodes in it.

Figure 2 introduces the operational semantics of an SXQ expression α with at most k free variables using a function $[\![\alpha]\!]_k$ that takes a data forest $\mathcal{F}$ and a $k-$tuple of nodes from the forest as input and returns an indexed forest. The input $k-$tuple of nodes represents an assignment of nodes to a given $k-$variables, that is, each variable $\$x_i$ is pointing to the node n_i, $1 \leq i \leq k$. The differences of the semantics of SXQ with respect to the semantics of XQ in [3] are:

- There is no rule for the constructor *lab*.
- There is no rule for the empty query represented by ().
- There is a new rule for the query represented by a sequence of variables inside a tag.

$$
\begin{array}{lrl}
\mathbf{XQ_1} & [\![\alpha\ \beta]\!]_k(\mathcal{F},\overline{e}) := & [\![\alpha]\!]_k(\mathcal{F},\overline{e}) \uplus [\![\beta]\!]_k(\mathcal{F},\overline{e}) \\
\mathbf{XQ_2} & [\![\text{for } \$x_{k+1} \text{ in } \alpha \text{ return } \beta]\!]_k(\mathcal{F},\overline{e}) := & \text{let } (\mathcal{F}',\bar{l}) = [\![\alpha]\!]_k(\mathcal{F},\overline{e}) \text{ in} \\
 & & \quad \biguplus_{1\leq i\leq|\bar{l}|} [\![\beta]\!]_{k+1}(\mathcal{F}',\overline{e}\cdot \bar{l}_i) \\
\mathbf{XQ_3} & [\![\$x_i]\!]_k(\mathcal{F},t_1,\ldots,t_k) := & (\mathcal{F},[t_i]) \\
\mathbf{XQ_4} & [\![\$x_i/\chi :: \nu]\!]_k(\mathcal{F},t_1,\ldots,t_k) := & (\mathcal{F}, \text{list of nodes } \upsilon \text{ such that } \chi^{\mathcal{F}}(t_i,\upsilon) \text{ and} \\
 & & \text{node } \upsilon \text{ has label } \nu \text{ in order } <^{tree(t_i)}_{doc}) \\
\mathbf{XQ_5} & [\![\text{if } \phi \text{ then } \alpha]\!]_k(\mathcal{F},\overline{e}) := & \text{if } \pi_2([\![\phi]\!]_k(\mathcal{F},\overline{e})) \neq [\ ] \text{ then } [\![\alpha]\!]_k(\mathcal{F},\overline{e}) \\
 & & \text{else } (\mathcal{F},[\ ]) \\
\mathbf{XQ_6} & [\![\text{if } \$x_i = \$x_j \text{ then } \alpha]\!]_k(\mathcal{F},t_1,\ldots,t_k) := & \text{if } t_i = t_j \text{ then } [\![\alpha]\!]_k(\mathcal{F},t_1,\ldots,t_k) \\
 & & \text{else } (\mathcal{F},[\ ]) \\
\mathbf{XQ_7} & [\![\langle a\rangle\ \langle/a\rangle]\!]_k(\mathcal{F},\overline{e}) := & \mathit{construct}(a,(\mathcal{F},[\ ])) \\
\mathbf{XQ_8} & [\![\langle a\rangle\ tag\ \langle/a\rangle]\!]_k(\mathcal{F},\overline{e}) := & \mathit{construct}(a,[\![tag]\!]_k(\mathcal{F},\overline{e})) \\
\mathbf{XQ_9} & [\![\langle a\rangle\ \$x_i\ \ldots\ \$x_j\ \langle/a\rangle]\!]_k(\mathcal{F},t_1,\ldots,t_k) := & \mathit{construct}(a,(\mathcal{F},[t_i,\ldots,t_j]))
\end{array}
$$

Fig. 2. Semantics of SXQ

This semantics makes use of some functions that construct indexed forest. The operator *construct*$(a,\ (\mathcal{F},\ [w_1...w_n]))$, denotes the construction of a new tree, where a is a label, $\mathcal{F}$ is a data forest, and $[w_1 \ldots w_n]$ is a list of nodes in $\mathcal{F}$. When applied, *construct* returns an indexed forest $(\mathcal{F} \cup T', [root(T')])$, where T' is a data tree with domain a new set of nodes, whose root is labeled with a, and with the subtree rooted at the i-th (in sibling order) child of root(T') being an isomorphic copy of the subtree rooted by w_i in $\mathcal{F}$. The symbol $\biguplus$ used in the rules takes two indexed forests $(\mathcal{F}_1,\bar{l}_1),(\mathcal{F}_2,\bar{l}_2)$ where the $\mathcal{F}_i$ are a data forest and $\bar{l}_i$ are lists of nodes in $\mathcal{F}_i$, and returns an indexed forest $(\mathcal{F}_1 \cup \mathcal{F}_2, \bar{l})$, where $\bar{l}$ is the concatenation of $\bar{l}_1$ and $\bar{l}_2$.

For a data tree $\mathcal{F}$, we let the binary relation $<^{\mathcal{F}}_{doc}$ on nodes be the *document-order* on $\mathcal{F}$: the depth-first left-to-right traversal order through $\mathcal{F}$. In the semantics of $\$x_i/\chi :: \nu$ we use $tree(t_i)$ to denote the maximal tree within the input forest that contains the node t_i, hence $<^{tree(t_i)}_{doc}$ is the document-order on the tree containing t_i. $\chi^{\mathcal{F}}$ is the interpretation of the axis relation of the same name in the data forest.

These semantic rules constitute a term rewriting system (TRS in short, see [2]), with each rule defining a single reduction step. The symbol $:=^*$ represents the reflexive and transitive closure of $:=$ as usual. The TRS is terminating and confluent (the rules are not overlapping).

As explained in [3], this semantics does not model the document() function of XQuery. Instead, we assume that there exist one or more initial variables that are each bound to a node of the input forest.

Given a SXQ query Q with $\mathit{free}(Q) = \{\$x_1, \ldots, \$x_k\}$, the semantics evaluates a query Q starting with the expression $[\![Q]\!]_k(\mathcal{F}, t_1, \ldots, t_k)$. The initial data forest $\mathcal{F}$ is a forest containing k input XML documents represented as data trees as explained in [3]. Each variable $\$x_i$ in $\mathit{free}(Q)$ represents an initial XML document and it is bound to a node of the input data forest $\mathcal{F}$. The sequence of nodes $t_1, \ldots, t_k$ from $\mathcal{F}$, corresponds to the nodes assigned to the variables $\{\$x_1, \ldots, \$x_k\}$. Along intermediate steps, expressions of the form $[\![Q']\!]_{k+n}(\mathcal{F}', t_1, \ldots, t_k, t_{k+1}, \ldots, t_{k+n})$ are obtained. The idea is that Q' is the subquery that can be found in Q at some position $p \in Pos(Q)$, and the set $Rel(Q, p)$ contains $k + n$ variables. The data forest $\mathcal{F}'$ is built from the input data forest $\mathcal{F}$ by adding (possible) new data trees, which are constructed by the operator *construct* representing new XML fragments.

The evaluation of a query returns as a result an indexed forest as a pair of the form $(\mathcal{F}', [e_1, \ldots, e_m])$ meaning that the query returns a sequence of $m-$nodes from $\mathcal{F}'$ representing XML fragments.

A more detailed discussion about this semantics and its properties can be found in [3].

3 $\mathcal{TOY}$ and Its Semantics

A $\mathcal{TOY}$ [14] program is composed of data type declarations, type alias, infix operators, function type declarations and defining rules for functions symbols. The syntax of (total)*expressions* in $\mathcal{TOY}$ $e \in Exp$ is $e ::= X \mid h \mid (e\ e')$ where X is a variable and h either a function symbol or a data constructor. Expressions of the form $(e\ e')$ stand for the application of expression e (acting as a function) to expression e' (acting as an argument). Similarly, the syntax of (total)*patterns* $t \in Pat \subset Exp$ can be defined as $t ::= X \mid c\ t_1 \ldots t_m \mid f\ t_1 \ldots t_m$ where X represents a variable, c a data constructor of arity greater or equal to m, and f a function symbol of arity greater than m, while the t_i are patterns for all $1 \leq i \leq m$. The set of *partial expressions* $Exp_\perp$ is the result of incorporating the new constant (0-arity constructor) $\perp$ to Exp. This constant plays the role of the undefined value. Similarly, the set of *partial patterns* $Pat_\perp$ is the result of incorporating the constant $\perp$ to Pat.

Data type declarations and type alias are useful for representing XML documents in $\mathcal{TOY}$:

```
data node      = txt    string
               | comment string
               | tag     string [attribute] [node]
data attribute = att     string string
type xml       = node
```

The data type `node` represents nodes in a simple XML document. It distinguishes three types of nodes: texts, tags (element nodes), and comments, each one represented by a suitable data constructor and with arguments representing the information about the node. For instance, constructor `tag` includes the tag name (an argument of type `string`) followed by a list of attributes, and finally a list of child nodes. The data type `attribute` contains the name of the attribute and its value (both of type `string`). The last type alias, `xml`, renames the data type `node`. Of course, this list is not exhaustive, since it misses several types of XML nodes, but it is enough for this presentation.
Each rule for a function f in $\mathcal{TOY}$ has the form:

$$\underbrace{f\ t_1 \ldots t_n}_{\text{left-hand side}} \rightarrow \underbrace{r}_{\text{right-hand side}} \Leftarrow \underbrace{e_1, \ldots, e_k}_{\text{condition}} \text{ where } \underbrace{s_1 = u_1, \ldots, s_m = u_m}_{\text{local definitions}}$$

where u_i and r are expressions (that can contain new extra variables) and t_i, s_i are patterns.
In $\mathcal{TOY}$, variable names must start with either an uppercase letter or an underscore (for anonymous variables), whereas other identifiers start with lowercase. $\mathcal{TOY}$ includes two primitives for loading and saving XML documents, called `load_xml_file` and `write_xml_file` respectively. For convenience, primitive `load_xml_file` includes a dummy tag `"root"` at the outer level. This is useful for grouping several XML fragments. If the file contains only one node `N` at the outer level, the `root` node is unnecessary, and can be removed using this simple function:

```
load_doc F = N <== load_xml_file F == xmlTag "root" [] [N]
```

where `F` is the name of the file containing the document. Observe that the strict equality `==` in the condition forces the evaluation of `load_xml_file F` and succeeds if the result has the form `xmlTag "root" [] [N]` for some `N`. If this is the case, `N` is returned.

The constructor-based ReWriting Logic (CRWL) [15] has been proposed as a suitable declarative semantics for functional-logic programming with lazy non-deterministic functions. The calculus is defined by five inference rules (see Figure 3): (BT) that indicates that any expression can be approximated by bottom, (RR) that establishes the reflexivity over variables, the decomposition rule (DC),

$$
\begin{array}{lll}
\textbf{BT} & e \rightarrow \perp & \\
\textbf{RR} & X \rightarrow X & \text{with } X \in Var \\
\textbf{DC} & \dfrac{e_1 \rightarrow t_1 \ \ldots \ e_m \rightarrow t_m}{h\ \overline{e}_m \rightarrow h\ \overline{t}_m} & h\ \overline{t}_m \in Pat_\perp \\
\textbf{JN} & \dfrac{e \rightarrow t \quad e' \rightarrow t}{e\ ==\ e'} & t \in Pat \text{ (total pattern)} \\
\textbf{FA} & \dfrac{e_1 \rightarrow t_1 \ \ldots \ e_n \rightarrow t_n \quad C \quad r\ \overline{a}_k \rightarrow t}{f\ \overline{e}_n\ \overline{a}_k\ \rightarrow t} & \\
 & \text{if } (f\ \overline{t}_n \rightarrow r \Leftarrow C) \in [P]_\perp, t \neq \perp &
\end{array}
$$

Fig. 3. CRWL Semantic Calculus

the (JN) (join) rule that indicates how to prove strict equalities, and the function application rule (FA). In every inference rule, $e, e_i \in Exp_\perp$ are partial expressions and $t_i, t, s \in Pat_\perp$ are partial patterns. The notation $[P]_\perp$ in the inference rule *FA* represents the set $\{(l \rightarrow r \Leftarrow C)\theta \mid (l \rightarrow r \Leftarrow C) \in P,\ \theta \in Subst_\perp\}$ of partial instances of the rules in the program P. The most complex inference rule is *FA* (Function Application), which formalizes the steps for computing a *partial pattern* t as approximation of a function call $f\ \overline{e}_n$:

1. Obtain partial patterns t_i as suitable approximations of the arguments e_i.
2. Apply a program rule $(f\ \overline{t}_n \rightarrow r \Leftarrow C) \in [P]_\perp$, verify the condition C, and check that t approximates the right-hand side r.

In this semantic notation, local declarations $a = b$ introduced in $\mathcal{TOY}$ syntax by the reserved word `where` are represented as part of the condition C as *approximation statements* of the form $b \rightarrow a$.

The semantics in $\mathcal{TOY}$ allows introducing non-deterministic functions, such as the following function `member` that returns all the elements in a list:

```
member:: [A]  -> A
member  [X | Xs] =  X
member  [X | Xs] =  member Xs
```

Another example of $\mathcal{TOY}$ function is the definition of the infix operator `.::.` which corresponds to the function composition:

```
infixr 90 .::.
(.::.) :: (A -> B) -> (B -> C) -> (A -> C)
(F .::. G) X = G (F X)
```

As the examples show, $\mathcal{TOY}$ is a typed language. However, the type declaration is optional and in the rest of the report they are omitted for the sake of simplicity. *Goals* in $\mathcal{TOY}$ are sequences of strict equalities. A strict equality

$e_1 == e_2$ holds (inference JN) if both e_1 and e_2 can be reduced to the same total pattern t. For instance, the goal `member [1,2,3,4] == R` yields four answers, the four values for `R` that make the equality true: $\{\texttt{R} \mapsto \texttt{1}\}, \ldots, \{\texttt{R} \mapsto \texttt{4}\}$.

The next lemma presents some easy consequences of the inference rules that are used in the proof of the main theoretical results.

Lemma 4. *Let t_1, t_2 be patterns and e be an expression. Then*

1. *If $\mathcal{P} \vdash t_1 \to t_2$, and t_2 is total, then $t_1 \equiv t_2$ (the symbol $\equiv$ is used to represent syntactic equivalence).*
2. *If $\mathcal{P} \vdash t_1 == t_2$, then $t_1 \equiv t_2$.*
3. *$\mathcal{P} \vdash e == t_1$, iff $\mathcal{P} \vdash e \to t_1$ and t_1 total.*
4. *It is always possible to prove $\mathcal{P} \vdash t_1 \to t_1$.*

Proof.

1. By structural induction on t_2. First observe that t_2 cannot contain $\perp$ because it is total, and that therefore the inference BT is never applied. If t_2 is a variable X, then the only inference applicable is RR and t_1 is also X. If $t_2 = h\ \overline{s'}_n$ for some patterns s'_i, then the only possible inference is DC, which implies that $t_1 = h\ \overline{s}_n$, and the result follows applying the inductive hypothesis to the premises.
2. The first step of the proof must consists of a JN inference rule. Thus, there is some total pattern t such that $\mathcal{P} \vdash t_1 \to t$, $\mathcal{P} \vdash t_2 \to t$. Then from the previous item, $t_1 \equiv t$, $t_2 \equiv t$, and therefore $t_1 \equiv t_2$.
3. First, assume $\mathcal{P} \vdash e == t_1$. In the premises of the JN we find $\mathcal{P} \vdash t_1 \to t$ for some total patter t. Then from the first item $t_1 \equiv t$. The other premise of the JN inference is $\mathcal{P} \vdash e \to t$, that is $\mathcal{P} \vdash e \to t_1$. Now suppose that $\mathcal{P} \vdash e \to t_1$ and t_1 total. Then we can prove $\mathcal{P} \vdash e == t_1$ taking $t \equiv t_1$ as the total pattern required by the JN inference.
4. If $t_1 \equiv \perp$ then the proof consists of a BT inference step, if it is a variable of a RR step, and if it is of the form $t_1 \equiv c\ \overline{s}_n$ of a DC step with premises $s_i \to s_i$ that can be proven in CRWL by induction hypothesis.

4 Transforming SXQ into $\mathcal{TOY}$

In order to represent SXQ queries in $\mathcal{TOY}$ we use some auxiliary datatypes:

```
type xPath = xml-> xml

data sxq  = xfor xml sxq sxq | xif cond sxq | xmlExp xml |
                     xp path | comp sxq sxq
data cond = sxq := sxq | cond sxq
data path = var xml | xml :/ xPath | doc string xPath
```

The structure of the datatype `sxq` allows representing any SXQ query (see Figure 4). It is worth noticing that a variable introduced by a *for* statement has type `xml`, indicating that the variable always contains a value of this type.

SXQ query	SXQ query in $\mathcal{TOY}$
query query	`comp sxq sxq`
for *var* **in** *query* **return** *query*	`xfor xml sxq sxq`
if *cond* **then** *query*	`xif cond sxq`
var	`xp (var xml)`
tag	`xmlExp xml`
$var/axis :: \nu$	`xp (xml :/ xPath)`
$var = var$	`sxq := sxq`

Fig. 4. Representation of SXQ queries in Figure 1 as a $\mathcal{TOY}$ terms

$\mathcal{TOY}$ includes a primitive `parse_xquery` that translates any SXQ expression into its corresponding representation as a term of this datatype, as the next example shows:

Example 6. The translation of the SXQ query of Example 2 into the datatype `sxq` produces the following $\mathcal{TOY}$ dataterm:

```
Toy> parse_xquery  "for $x3 in $x1/child::bib return
           for $x4 in .....  <rev> $x9  $x10 </rev>" == R
yes
{R --> xfor X3 (xp ( X1 :/ (child .::. (nameT "bib"))))
    (xfor X4 (xp ( X3 :/ (child .::.(nameT "book"))))
    (xfor X5 (xp ( X2 :/ (child .::.(nameT "reviews"))))
    (xfor X6 (xp ( X5 :/ (child .::.(nameT "entry"))))
    (xfor X7 (xp ( X4 :/ (child .::.(nameT "title"))))
    (xfor X8 (xp ( X6 :/ (child .::.(nameT "title"))))
    (xif  ( xp ( var X7 ) :=  xp ( var X8) )
    (xfor X9 (xp ( X6 :/ (child .::.(nameT "title"))))
    (xfor X10 (xp ( X6 :/ (child .::.(nameT "review"))))
       (xmlExp (xmlTag "rev" [] [X9, X10]))))))))))
 }
```

The primitive `parse_xquery` takes as input a SXQ expression Q and returns as ouput the query `Q` represented as a $\mathcal{TOY}$ dataterm.

Without loss of generality, in order to simplify our implementation, the primitive `parse_xquery` also allows as input queries without free variables. This is possible by replacing all the free variables in the SXQ query Q by its corresponding XML files. That is, in Example 6 instead of variables $\$x1$ and $\$x2$ we have the strings "doc(bib.xml)" and "doc(reviews.xml)" respectively.

The interpreter assumes the existence of the infix operator `.::.` that connects axes and tests to build steps (the operator `::` in XPath syntax), defined as the sequence of applications in Chapter 3.

The rules of the $\mathcal{TOY}$ interpreter that processes SXQ queries can be found in Figure 5. The main function is `sxq`, which distinguishes cases depending of the form of the query. If it is an XPath expression then the auxiliary function

```
sxq (xp E)             = sxqPath E
sxq (xmlExp X)         = X
sxq (comp Q1 Q2)       = sxq Q1
sxq (comp Q1 Q2)       = sxq Q2
sxq (xfor X Q1 Q2)     = sxq Q2 <== X== sxq Q1
sxq (xif (Q1:=Q2) Q3)  = sxq Q3 <== sxq Q1  == sxq Q2
sxq (xif (cond Q1) Q2) = sxq Q2 <== sxq Q1  == _

sxqPath (var X)  = X
sxqPath (X :/ S) = S X
sxqPath (doc F S) = S (load_xml_file F)

%%%% XPATH %%%%%%
attr A (xmlTag S Attr L ) = xmlText T <== member Attr == xmlAtt A T
nameT S (xmlTag S Attr L ) = xmlTag S Attr L
nodeT X = X
textT (xmlText S) = xmlText S
commentT S (xmlComment S) = xmlComment S

self X = X
child (xmlTag _Name _Attr L) = member L
descendant X = child X
descendant X = descendant Y <== child X == Y
dos = self
dos = descendant
```

Fig. 5. $\mathcal{TOY}$ transformation rules for SXQ

sxqPath is used. If the query is an XML expression, the expression is just returned (this is safe thanks to our constraint of allowing only variables inside XML expressions). If we have two queries (comp construct), the result of evaluating any of them is returned using non-determinism. The for statement (xfor construct) forces the evaluation of the query Q1 and binds the variable X to the result. Then the result query Q2 is evaluated. The case of the if statement is analogous. The XPath subset considered includes tests for attributes (attr), label names (nameT), general elements (nodeT), text nodes (textT) and comments (commentT). It also includes the axes self, child, descendant and *descendant or self* (called here dos as usual in XQuery). Observe that we do not include reverse axes like ancestor because they can be replaced by expressions including forward axes, as shown in [16,4]. Other constructions such as filters can be easily included (see [4]). The next example uses the interpreter to obtain the answers for the query of our running example.

Example 7. The goal:

```
Toy> sxq (parse_xquery "for....") == R
```

applies the interpreter of Figure 5 to the code of Example 6 (assuming that the string after `parse_xquery` is the query in Example 2), and returns the $\mathcal{TOY}$ representation of the expected results:

```
<rev>
 <title>TCP/IP Illustrated</title>
 <review>  One of the best books on TCP/IP.  </review>
</rev>
...
```

Regarding performance, the current main limitation is that the primitive `load_xml_file` cannot load documents with size beyond a few megabytes. Our experiments with these medium-size files indicate that the interpreter computes the answer in a reasonable amount of time, even for complex queries.

4.1 Soundness of the Transformation

One of the goals of this report is to ensure that the embedding is semantically correct and complete. This section introduces the theoretical results establishing these properties. If V is a set of indexed variables of the form $\{X_1, \ldots, X_n\}$ and θ a substitution on these variables, we use the notation $\theta(V)$ to indicate the sequence $\theta(X_1), \ldots, \theta(X_n)$. In the following results it is implicitly assumed that there is a bijective mapping f from XML format to the datatype `xml` in $\mathcal{TOY}$. However, in order to simplify the presentation, we omit the explicit mention to f and to its inverse f^{-1}. Also, variables in SXQ queries, with names of the form $\$x_i$ are assumed to be represented in $\mathcal{TOY}$ as X_i and conversely.

Lemma 5. *Let P be a $\mathcal{TOY}$ program, Q' be a SXQ query and* `Q'` *the representation of Q as a $\mathcal{TOY}$ dataterm. Let $p \in Pos(Q')$ be a position of the query Q' such that $Q'_{|p} \equiv Q$ with $Rel(Q',p) = \{\$x_1, \ldots, \$x_k\}$ (see Definition 5). Let θ be a substitution such that $dom(\theta) = Rel(Q',p)$ and $\mathcal{P} \vdash (\texttt{sxq Q}\theta == \texttt{t})$ for some pattern* `t`.
Then, for every data forest $\mathcal{F}$, containing the list of nodes $t_1, \ldots, t_k$ with $t_1 = \theta(\$x_1), \ldots, t_k = \theta(\$x_k)$, there exists an indexed forest $(\mathcal{F}', L')$ such that:

$$[\![Q]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}', L')$$

verifying $t \in L'$.

Proof. Observe that from $\mathcal{P} \vdash (\texttt{sxq Q}\theta == \texttt{t})$ and by Lemma 4, item 3 we have $\mathcal{P} \vdash (\texttt{sxq Q}\theta \to \texttt{t})$. Suppose $\theta = \{x_1 \mapsto t_1, \ldots, x_k \mapsto t_k\}$. We prove by complete induction on the structure of Q that if $\mathcal{P} \vdash (\texttt{sxq Q}\theta \to \texttt{t})$ then for all data forest $\mathcal{F}$ containing the list of nodes $t_1, \ldots, t_k$, there exists an indexed forest $(\mathcal{F}', L')$ such that: $[\![Q]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}', L')$, verifying $t \in L'$.

- $Q \equiv Q_1\ Q_2$. The query Q is represented in $\mathcal{TOY}$ as `comp Q1 Q2`. Any proof of $\mathcal{P} \vdash \texttt{sxq (comp Q1 Q2)}\theta \to \texttt{t}$ must start by a (FA) CRWL reduction step

(see Figure 3), which must use an instance either of the rule `sxq (comp Q1 Q2) = sxq Q1` or of the rule `sxq (comp Q1 Q2) = sxq Q2`. Assume the first rule is used (analogous for the second one).
Applying the rule instance `sxq (comp Q1' Q2') = sxq Q1'`, the proof of $\mathcal{P} \vdash \texttt{sxq (comp Q1 Q2)}\theta \to \texttt{t}$ is of the form:

$$\frac{\texttt{(comp (Q1 Q2))}\theta \to \texttt{(comp (Q1' Q2'))}\sigma \qquad \texttt{sxq Q1'}\sigma \to \texttt{t}}{\texttt{sxq (comp (Q1 Q2))}\theta \ \to \ \texttt{t}}$$

with $\sigma = \{\texttt{Q1}' \mapsto \texttt{Q1}, \texttt{Q2}' \mapsto \texttt{Q2}\} \cdot \theta$. Then the (FA) inference step has a premise proving $\mathcal{P} \vdash \texttt{sxq Q1}\theta \to t$.
We check that the induction hypothesis can be applied to Q_1 verifying that it satisfies the premises of the lemma.

- $Q_1 \equiv Q_{|1}$ then $Q_1 \equiv Q'_{|p \cdot 1}$ is an SXQ query. Note that, $p \cdot 1 \in Pos(Q')$.
- By Lemma 3, $Rel(Q', p \cdot 1) = Rel(Q', p)$.

In SXQ : By $\mathbf{XQ_1}$

$$(1) \qquad [\![Q]\!]_k(\mathcal{F}, t_1, \ldots, t_k) := [\![Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k) \ \uplus \ [\![Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k)$$

By the induction hypothesis:

$$(2) \qquad [\![Q_1]\!]_k(\mathcal{F},\, t_1, \ldots, t_k) :=^* (\mathcal{F}_1,\, L_1)$$

with $\mathcal{F}_1$ some forest containing the nodes in the list L_1, verifying $t \in L_1$.
Combining (1) and (2) and considering that := is normalizing, that is:

$$[\![Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}_2, L_2)$$

for some indexed forest $(\mathcal{F}_2, L_2)$. Then:

$$[\![Q]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}_1 \cup \mathcal{F}_2,\, L_1 +\!\!+ L_2)$$

with $t \in L_1$ and thus $t \in L_1 +\!\!+ L_2$.

– $Q \equiv \textsf{for}\ \$x_{k+1}\ \textsf{in}\ \ Q_1\ \textsf{return}\ \ Q_2$. This query introduces a new variable by means of a *for* statement.
The query Q is represented in $\mathcal{TOY}$ as `xfor` $\texttt{X}_{k+1}$ `Q1 Q2`. Then any proof of $\mathcal{P} \vdash \texttt{sxq Q}\theta \to \texttt{t}$ must start with a (FA) inference using a variant of the program rule: `sxq (xfor X Q1' Q2') = sxq Q2' <== X== sxq Q1'`.

$$\frac{\begin{array}{ll}(1) & \texttt{(xfor X}_{k+1}\texttt{ Q1 Q2)}\theta \to \texttt{(xfor X Q1' Q2')}\sigma \\ (2) & \texttt{sxq Q1'}\sigma \ == \ \texttt{X}\sigma \\ (3) & \texttt{sxq Q2'}\sigma \to \ \texttt{t}\end{array}}{\texttt{(sxq (xfor X}_{k+1}\texttt{ Q1 Q2))}\theta \ \to \ \texttt{t}}$$

with $\sigma = \{\texttt{X} \mapsto \texttt{X}_{\texttt{k+1}}, \texttt{Q1}' \mapsto \texttt{Q1}, \texttt{Q2}' \mapsto \texttt{Q2}\} \cdot \theta$. This proof can be rewritten as:

$$\frac{\begin{array}{ll}(1) & \texttt{(xfor X}_{k+1}\texttt{ Q1 Q2)}\theta \to \texttt{(xfor X Q1' Q2')}\sigma \\ (2) & \texttt{sxq Q1}\theta \ == \ \texttt{X}_{k+1}\theta \\ (3) & \texttt{sxq Q2}\theta \to \ \texttt{t}\end{array}}{\texttt{(sxq (xfor X}_{k+1}\texttt{ Q1 Q2))}\theta \ \to \ \texttt{t}}$$

There is a CRWL proof for the three premises. The strict equality `sxq Q1`θ `==` $\mathtt{X}_{k+1}\theta$ holds (inference *JN*); there is a term `t'` such that both `sxq Q1`θ and $\mathtt{X}_{k+1}\theta$ can be reduced to `t'`. The proof must be of the form:

$$\frac{\begin{array}{l}(\mathtt{sxq\ Q1})\theta' \rightarrow \mathtt{t'} \\ \mathtt{X}_{k+1}\theta' \rightarrow \ \mathtt{t'}\end{array}}{\mathtt{sxq\ Q1}\theta\ \ \mathtt{==}\ \ \mathtt{X}_{k+1}\theta}$$

with $\theta' = \{\mathtt{X_{k+1}} \mapsto \mathtt{t'}\} \cdot \theta$.
Next we check that Q_1 and Q_2 verify the lemma premises, and that hence it is possible to apply the induction hypothesis to Q_1 and Q_2.

In the case of Q_1:

- $Q_1 \equiv Q'_{|p\cdot 1}$.
- By Lemma 3, $Rel(Q', p \cdot 1) = Rel(Q', p)$.

Then applying induction:

$$[\![Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}_1, L_1)$$

with $\mathcal{F}_1$ some data forest containing all the nodes in L_1 and verifying:

$$t' \in L_1 \tag{3}$$

Notice that for all data tree $\mathcal{T} \in \mathcal{F}$, $\mathcal{T} \in \mathcal{F}_1$. Then it is posible to ensure that the data forest $\mathcal{F}_1$ contains all the nodes in the list $\{t_1, \ldots, t_k\}$.

In the case of Q_2:

- $Q_2 \equiv Q'_{|p\cdot 2}$.
- By Lemma 3, $Rel(Q', p \cdot 2) = Rel(Q', p) \cup \{\$x_{k+1}\}$.
- Additionally, in the premises of the CRWL proof there is a CRWL proof for $\mathcal{P} \vdash$ `sxq Q2`$\theta \rightarrow$ `t`. The proof must use a substitution of the form $\theta' = \{\mathtt{X_{k+1}} \mapsto \mathtt{t'}\}\cdot\theta$. Then, there is a CRWL proof for $\mathcal{P} \vdash$ `sxq Q2`$\theta' \rightarrow$ `t`.

Then applying induction:

$$[\![Q_2]\!]_{k+1}(\mathcal{F}_1, t_1, \ldots, t_k, t') :=^* (\mathcal{F}_2, L_2), \quad t \in L_2 \tag{4}$$

In SXQ , by $\mathbf{XQ_2}$:

$[\![Q]\!]_k(\mathcal{F}, t_1, \ldots, t_k) := \biguplus_{1\leq j\leq |L_1|} [\![Q_2]\!]_{k+1}(\mathcal{F}_1, t_1, \ldots, t_k, l_j) = (\mathcal{F}', L')$
with:
$[\![Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}_1, L_1)$, with l_j the l_j-th node in L_1.
Then from (3) t' is one of these l_j, and from (4) $t \in L_2$, and thus $t \in L'$.

– $Q \equiv \$x_i$. The query Q is such that $Q'_{|p} \equiv Q$ for $p \in Pos(Q')$. By Definition 1, $\mathit{free}(Q'_{|p}) = \{\$x_i\}$ and by Lemma 2, $\$x_i \in Rel(Q', p) = \{\$x_1, \ldots, \$x_k\}$. The representation of this query in $\mathcal{TOY}$ will be `xp (var` $\mathtt{X}_i$`)`. Any proof for $\mathcal{P} \vdash$ `sxq (xp (var` $\mathtt{X}_i$`))`$\theta \rightarrow$ `t` must start with a (FA) inference using a variant of the program rule `sxq (xp E) = sxqPath E`. Therefore this inference has a premise proving $\mathcal{P} \vdash$ `sxqPath (var` $\mathtt{X}_i)\theta \rightarrow t$.

This proof must use again the (FA) inference, this time applying the rule `sxqPath (var X) = X`. Therefore in the proof of this statement we find a proof for $\mathcal{P} \vdash \mathtt{X}_i\theta \to t$, which by Lemma 4 implies that $\theta(X_i) \equiv t$.
In SXQ . Applying $\mathbf{XQ_3}$,

$$[\![\$x_i]\!]_k(\mathcal{F}, t_1, \ldots, t_k) := (\mathcal{F}, [t_i])$$

with $\theta(x_i) = t_i$. Then $t_i = t$ and the result holds.

– $Q \equiv \$x_i/axis :: \nu$. The query Q is such that $Q'_{|p} \equiv Q$ for $p \in Pos(Q')$. By Definition 1, $free(Q'_{|p}) = \{\$x_i\}$ and by Lemma 2, $\$x_i \in Rel(Q', p) = \{\$x_1, \ldots, \$x_k\}$.
We check the case where the axis is `child` and the test a node name (the proof is analogous for the rest of axes and tests). In this case the representation in $\mathcal{TOY}$ of the query Q is: `xp` $\mathtt{X}_i$ `:/ child .::. (nameT "name")`.
From the premise $\mathcal{P} \vdash \mathtt{sxq\ Q}\theta == \mathtt{t}$ and by Lemma 4 there is a CRWL proof for $\mathcal{P} \vdash \mathtt{sxq\ Q}\theta \to \mathtt{t}$.
The proof must start applying a (FA) inference rule of CRWL, applying the rule `sxq (xp E) = sxqPath E` (see Figure 5). The step must be of the form:

$$\frac{\begin{array}{l}\mathtt{X}_i\theta\ \mathtt{:/\ child\ .::.\ (nameT}\ \mathit{name}\mathtt{)} \to \mathtt{X}_i\theta\ \mathtt{:/\ child\ .::.\ (nameT}\ \mathit{name}\mathtt{)} \\ \mathtt{sxqPath\ (X}_i\theta\ \mathtt{:/\ child\ .::.\ (nameT}\ \mathit{name}\mathtt{))} \to \mathtt{t}\end{array}}{\mathtt{sxq\ Q}\theta \to \mathtt{t}}$$

with $\sigma = \{E \mapsto \mathtt{X}_i$ `:/ child .::. (nameT "name")`$\}\theta$.
The second premise must have a proof starting with a (FA) inference applying the second rule of `sxqPath` (that is, `sxqPath (X :/ S) = S X`, see Figure 5), and with an instance given by the substitution $\sigma = \{\mathtt{X} \mapsto \mathtt{X}_i\theta,\ S \mapsto$ `child .::. (nameT` *name*`)`$\}$:

$$\frac{\begin{array}{l}\mathtt{X}_i\theta \to \mathtt{X}_i\theta \\ \mathtt{child\ .::.\ (nameT}\ \mathit{name}\mathtt{)} \to \mathtt{child\ .::.\ (nameT}\ \mathit{name}\mathtt{)} \\ \mathtt{(child\ .::.\ (nameT}\ \mathit{name}\mathtt{))\ X}_i\theta \to \mathtt{t}\end{array}}{\mathtt{sxqPath\ (X}_i\theta\ \mathtt{:/\ child\ .::.\ (nameT}\ \mathit{name}\mathtt{))} \to \mathtt{t}}$$

The two first premises correspond to the pattern matching of parameters. The third premise is the reduction of the right-hand side, and must apply once more an FA inference, this time using the rule for the infix operator `.::.`. The used rule is `(F .::. G) X = G (F X)`, see Section 3, with instance

$$\sigma = \{\mathtt{F} \mapsto \mathtt{child}, \mathtt{G} \mapsto \mathtt{nameT}\ \ \mathit{name}, \mathtt{X} \mapsto \mathtt{X}_i\theta\}$$

The inference must be of the form:

$$\frac{\begin{array}{l}\mathtt{child} \to \mathtt{child} \\ \mathtt{nameT}\ \mathit{name} \to \mathtt{nameT}\ \mathit{name} \\ \mathtt{X}_i\theta \to \mathtt{X}_i\theta \\ \mathtt{(nameT}\ \mathit{name}\mathtt{)\ (\ child\ X}_i\theta\mathtt{)} \to \mathtt{t}\end{array}}{\mathtt{(child\ .::.\ (nameT}\ \mathit{name}\mathtt{))\ X}_i\theta \to \mathtt{t}}$$

Now the proof of `(child .::. (nameT name))`$X_i\theta \to$ `t` corresponds to an application of function `nameT` with two arguments: *name* and `(child` $X_i\theta$`)`. The program rule for `nameT` is `nameT S (xmlTag S Attr L ) = xmlTag S Attr L`. In order to apply this rule the second argument of `nameT`, `(child` $X_i\theta$`)`, must be reduced to a pattern of the form `xmlTag name Attr L`. Therefore the substitution must be $\sigma = \{\mathtt{S} \mapsto \mathit{name}\}$. Then the FA step is of the form:

$$\frac{\begin{array}{l}\mathit{name} \to \mathit{name}\\ \mathtt{child}\ \mathtt{X}_i\theta \to \mathtt{xmlTag}\ \mathit{name}\ \mathtt{Attr}\ \mathtt{L}\\ \mathtt{xmlTag}\ \mathit{name}\ \mathtt{Attr}\ \mathtt{L} \to \mathtt{t}\end{array}}{(\mathtt{nameT}\ \mathit{name})\ (\mathtt{child}\ \mathtt{X}_i\theta) \to \mathtt{t}}$$

Since `t` is a total pattern, from Lemma 4 applied to the third premise we have $\mathtt{t} \equiv \mathtt{xmlTag}\ \mathit{name}\ \mathtt{Attr}\ \mathtt{L}$, that is, `t` is the representation in $\mathcal{TOY}$ of an XML element with label *name*. The second premise implies a proof for `child` $X_i\theta \to$ `xmlTag name Attr L` in CRWL. Again the rule FA is applied, this time using the program rule `child (xmlTag Name' Attr' L') = member L'` (the variables have been renamed). The instance must use a substitution of the form $\sigma = \{Name' \mapsto \mathtt{A}, Attr' \mapsto \mathtt{B}$ for some patterns `A` and `B`. The FA step must be of the form:

$$\frac{\begin{array}{l}\mathtt{X}_i\theta \to \mathtt{xmlTag\ A\ B\ L'}\\ \mathtt{member\ L'} \to \mathtt{xmlTag}\ \mathit{name}\ \mathtt{Attr\ L}\end{array}}{\mathtt{child}\ \mathtt{X}_i\theta \to \mathtt{xmlTag}\ \mathit{name}\ \mathtt{Attr\ L}}$$

It is easy to prove that `member L'` returns all the members in `L'` (by induction on the length of `L'`). Therefore:

1. $X_i\theta$ is a value of the form `xmlTag A B L'` for some values `A`, `B` and L'.
2. $\mathtt{t} \equiv \mathtt{xmlTag}\ \mathit{name}\ \mathtt{Attr\ L}$ is in L', which means that `t` is a child of $X_i\theta$ with label *name*.

In SXQ : Observe that $\$x_i \in Rel(Q', p)$. Applying $\mathbf{XQ_4}$ we have that for all data forest $\mathcal{F}$, containing the list of nodes $t_1, \ldots, t_k$,

$$[\![\$x_i/\text{child} :: \mathit{name}]\!]_k(\mathcal{F}, t_1, \ldots, t_k) = (\mathcal{F}, L'')$$

Then we prove that $t \in L''$. This holds because by $\mathbf{XQ_4}$, L'' is the list of nodes υ such that [1]:

i) $\text{child}^{\mathcal{F}}(t_i, \upsilon)$. The children of t_i = `xmlTag A B L'` are the elements of L'. Then $t \in L'$, and therefore it satisfies this condition.
ii) Label `name` of $(\upsilon) = \mathit{name}$. The label of t is *name*.

Therefore $t \in L''$ as indicated in the lemma.

The rest of the cases are analogous to the previous cases.

[1] The condition about the order in the nodes in $\mathbf{XQ_4}$ is not included because it has no effect in the result.

The theorem that establishes the correctness of the approach is an easy consequence of the previous Lemma.

Theorem 1. *Let P be the $\mathcal{TOY}$ program of Figure 5, Q a SXQ query with free$(Q) = \{\$x_1, \ldots, \$x_m\}$. Let* `Q` *be the representation of Q as a $\mathcal{TOY}$ dataterm according to the table in Figure 4,* `t` *be a $\mathcal{TOY}$ pattern, and θ a substitution such that $dom(\theta) = free(Q)$ and $\mathcal{P} \vdash (\texttt{sxq Q}\theta == \texttt{t})$. Then, for all data forest $\mathcal{F}$ containing the nodes $t_1, \ldots, t_m$ with $t_1 = \theta(x_1), \ldots, t_m = \theta(x_m)$, there exists an indexed forest $(\mathcal{F}', L')$ such that:*

$$[\![Q]\!]_m(\mathcal{F}, t_1, \ldots, t_m) :=^* (\mathcal{F}', L')$$

verifying $t \in L'$.

Proof. In Lemma 5 consider the position $p \equiv \varepsilon$. Then $Q' \equiv Q$, $Rel(Q, p) = free(Q) \cup V_{for}(Q, \varepsilon) = free(Q) \cup \emptyset = \{\$x_1, \ldots, \$x_m\}$. Then, the conclusion of the theorem is the conclusion of the lemma.

Thus, our approach is correct. The next Lemma allows us to prove that it is also complete, in the sense that the $\mathcal{TOY}$ program can produce every answer obtained by the SXQ operational semantics.

Lemma 6. *Let P be the $\mathcal{TOY}$ program of Figure 5. Let Q' be a SXQ query and p a position in $Pos(Q')$ such that $Q \equiv Q'_{|p}$ and $Rel(Q', p) = \{\$x_1, \ldots, \$x_k\}$. Suppose that $[\![Q]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}', L)$ for some $\mathcal{F}$, $\mathcal{F}'$, $t_1, \ldots, t_k$, L. Then, for every $t \in L$, there is a substitution θ such that $\theta(\$x_i) = t_i$ for all $\$x_i \in Rel(Q', p)$ and a CRWL-proof proving $\mathcal{P} \vdash \texttt{sxq Q}\theta == \texttt{t}$.*

Proof. Due to the Lemma 4 it is enough to prove that $\mathcal{P} \vdash \texttt{sxq Q}\theta \to \texttt{t}$ by complete induction on the structure of Q.

- $Q \equiv Q_1\ Q_2$.
  In this case, by $\mathbf{XQ_1}$,
  $$\begin{aligned}[\![Q_1 Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k) &:= [\![Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k) \uplus [\![Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k) \\ &:=^* (\mathcal{F}_1, L_1) \uplus (\mathcal{F}_2, L_2)\end{aligned}$$
  Then t is either in L_1 or in L_2.
  - If t in L_1. Then we consider the reduction $[\![Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}_1, L_1)$. The set of variables of Q' that are relevant for Q_1 is denoted by $Rel(Q', p\cdot 1)$, and by Lemma 3 $Rel(Q', p\cdot 1) = Rel(Q', p)$. For all $\$x_i$ in $Rel(Q', p\cdot 1)$, $\theta(\$x_i) = t_i$, and by induction hypothesis, for every $t \in L_1$, there is a CRWL-proof proving $\mathcal{P} \vdash \texttt{sxq Q1}\theta \to \texttt{t}$. Now, applying a variant of the the third rule of `sxq` (for instance, `sxq (comp Q1' Q2') = sxq Q1'`, see Figure 5), there is a CRWL proof for $\mathcal{P} \vdash \texttt{sxq (comp (Q1 Q2))}\theta \to \texttt{t}$ using the substitution $\sigma = \{\texttt{Q1}' \mapsto \texttt{Q1}, \texttt{Q2}' \mapsto \texttt{Q2}\} \cdot \theta$ to obtain the rule instance. The first step of the proof is:

$$\frac{\texttt{(comp (Q1 Q2))}\theta \to \texttt{(comp (Q1' Q2'))}\sigma \qquad \texttt{sxq Q1'}\sigma \to \texttt{t}}{\texttt{sxq (comp (Q1 Q2))}\theta \ \to\ \texttt{t}}$$

The CRWL-proof of the first premise is obtained from Lemma 4 since both sides are the same term due to the definition of σ. The second premise is the result we have obtained by induction hypothesis since `sxq Q1'`σ = `sxq Q1`θ.

- If t in L_2. Analogously to the previous case, the induction hypothesis can be applied to Q_2, concluding that for every $t \in L_2$, there is some CRWL-proof proving $\mathcal{P} \vdash$ `sxq Q2` $\theta \to$ `t`, and hence for $\mathcal{P} \vdash$ `sxq (comp (Q1 Q2))`$\theta \to$ `t` using the fourth rule of `sxq` (Figure 5).

In both cases for all $t \in L_1 \cup L_2$, there is a CRWL-proof proving

$$\mathcal{P} \vdash \texttt{sxq (comp (Q1 Q2))}\theta \to \texttt{t}$$

which proves the result.

– $Q \equiv \textsf{for}\ \ \$x_{k+1}\ \textsf{in}\ \ Q_1\ \textsf{return}\ \ Q_2$.
In this case, by $\mathbf{XQ_2}$,

$$\begin{aligned}\llbracket \text{for } \$x_{k+1} \text{ in } Q_1 \text{ return } Q_2 \rrbracket_k(\mathcal{F}, t_1, \ldots, t_k) &:= \biguplus_{1 \le i \le m} \llbracket Q_2 \rrbracket_{k+1}(\mathcal{F}', t_1, \ldots, t_k, l_i) \\ &:=^* (\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_m, L_1 + + \cdots + + L_m)\end{aligned}$$

where $\llbracket Q_1 \rrbracket_k(\mathcal{F}, t_1, \ldots, t_k) := (\mathcal{F}', [l_1, \ldots l_m])$.
If $L_1 + + \cdots + + L_m = [\ ]$ the result trivially holds. In other case, consider any $t \in L_1 + + \cdots + + L_m$
Now, we check the induction hypothesis can be applied to both Q_1 and Q_2.

- The set of variables of Q' that are relevant for Q_1 is denoted by $Rel(Q', p \cdot 1)$, and by Lemma 3, $Rel(Q', p) = Rel(Q', p \cdot 1)$.
By induction hypothesis there is a CRWL-proof proving $\mathcal{P} \vdash$ `sxq Q1` $\theta_1 \to l_r$ for some substitution θ_1 such that $\theta_1(\$x_i) = t_i$, for $t_i = 1 \ldots k$, with $\$x_i \in Rel(Q', p \cdot 1)$.
- Consider the reduction $\llbracket Q_2 \rrbracket_{k+1}(\mathcal{F}', t_1, \ldots, t_k, l_r) :=^* (\mathcal{F}_r, L_r)$. In this case $Rel(Q', p \cdot 2)$ be the set of variables of Q' that are relevant for Q_2. Then, by Lemma 3, $Rel(Q', p \cdot 2) = Rel(Q', p) \cup \{\$x_{k+1}\}$.
By induction hypothesis, for all $t \in L_r$, there is a CRWL-proof proving $\mathcal{P} \vdash$ `sxq Q2`$\theta_2 \to$ `t` for some θ_2 such that $\theta_2(\$x_j) = \theta_1(\$x_j) = t_j$, $1 \le j \le k$ and $\theta_2(\$x_{k+1}) = l_r$. Then we can define $\theta = \theta_1 \cup \theta_2$ without ambiguity, because $dom(\theta_1) \cap dom(\theta_2) = Rel(Q', p)$ and $\theta_2(\$x) = \theta_1(\$x)$ for every $\$x \in Rel(Q', p)$.
Now we can use a variant of the the fifth rule of `sxq` (see Figure 5) such as `sxq (xfor X Q1' Q2') = sxq Q2' <== X== sxq Q1'`, and a substitution $\sigma = \{\texttt{X} \mapsto \texttt{X}_{k+1}, \texttt{Q1}' \mapsto \texttt{Q1}, \texttt{Q2}' \mapsto \texttt{Q2}\} \cdot \theta$, and build a CRWL-proof for $\mathcal{P} \vdash$ (`sxq (xfor X`$_{k+1}$ `Q1 Q2)`)$\theta \to$ `t` starting with a (FA) inference of the form:

$$\frac{\begin{array}{l}(\texttt{xfor X}_{k+1}\ \texttt{Q1 Q2})\theta \to (\texttt{xfor X Q1' Q2'})\sigma \\ \texttt{sxq Q1'}\sigma\ ==\ \texttt{X}\sigma \\ (\texttt{sxq Q2'})\sigma \to \texttt{t}\end{array}}{(\texttt{sxq (xfor X}_{k+1}\ \texttt{Q1 Q2}))\theta \to \texttt{t}}$$

which can be rewritten as

$$\frac{\begin{array}{ll}(1) & (\texttt{xfor X}_{k+1}\texttt{ Q1 Q2})\theta_2 \rightarrow (\texttt{xfor X}_{k+1}\texttt{ Q1 Q2})\theta_2 \\ (2) & \texttt{sxq Q1}\theta_1 \texttt{ == X}_{k+1}\theta_2 \\ (3) & \texttt{sxq Q2}\theta_2 \rightarrow \texttt{ t}\end{array}}{(\texttt{sxq (xfor X}_{k+1}\texttt{Q1 Q2}))\theta_2 \rightarrow \texttt{ t}}$$

taking into account the definition of σ and θ.
Now we check that the three premises can be proven in CRWL.

1. $\mathcal{P} \vdash (\texttt{xfor}\ X_{k+1}\ Q_1\ Q_2)\theta_2 \rightarrow (\texttt{xfor}\ X_{k+1}\ Q_1\ Q_2)\theta_2$. Holds by Lemma 4.
2. $\mathcal{P} \vdash \texttt{sxq}\ Q_1\theta_1\ \texttt{==}\ X_{k+1}\theta_2$. Considering that $X_{k+1}\theta_2 = l_r$, and by Lemma 4, we must find a proof for $\mathcal{P} \vdash \texttt{sxq}\ Q_1\theta_1 \rightarrow l_r$, and such proof exists by induction hypothesis.
3. $\mathcal{P} \vdash \texttt{sxq}Q_2\theta_2 \rightarrow \texttt{t}$. Holds by induction hypothesis.

Observe that in fact θ_2 contains also $\$x_{k+1}$ in its domain, with $\$x_{k+1} \notin Rel(Q', p)$, but it still verifies the requirements of the Lemma, because $\theta_2(\$x_i) = t_i$ for $\$x_i \in Rel(Q', p)$.

– $Q \equiv \texttt{if}\ Q_1\texttt{then}\ Q_2$. In $\mathcal{TOY}$: `xif (cond Q1 ) Q2 .`
In this case, by $\mathbf{XQ_5}$,

$$[\![\text{if } Q_1 \text{ then } Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k) := \begin{array}{l}\text{if } \pi_2([\![Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k)) \neq [\,] \\ \text{then } [\![Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k) \\ \text{else } (\mathcal{F}, [\,])\end{array}$$

We distinguish two cases:

- $[\![\text{if } Q_1 \text{ then } Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}, [\,])$
In this case, $[\![\text{if } Q_1 \text{ then } Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k) = (\mathcal{F}, [\,])$. Therefore, the result trivially holds.
- $[\![\text{if } Q_1 \text{ then } Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k) := [\![Q_2]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}', L')$.
In this case, the condition Q_1 returns some result, that is, $[\![Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}'', L'')$. Let t be any value in L'. Then we prove that $\mathcal{P} \vdash (\texttt{sxq Q})\theta \rightarrow \texttt{t}$ for some θ with $\theta(\$x_i) = t_i$ for every $1 \leq i \leq k$. By Lemma 3, $Rel(Q', p) = Rel(Q', p \cdot 1)$ and $Rel(Q', p) = Rel(Q', p \cdot 2)$.
Hence the induction hypothesis can be applied to both Q_1 and Q_2.
  - ∗ For $t' \in L''$, there ia a substitution θ_1 such that $\theta_1(\$x_i) = t_i$ for $\$x_i \in Rel(Q', p)$, and a CRWL proof proving $\mathcal{P} \vdash (\texttt{sxq Q1})\theta_1 \rightarrow \texttt{t}'$.
  - ∗ For $t \in L'$, there is a a substitution θ_2 such that $\theta_2(\$x_i) = t_i$ for $\$x_i \in Rel(Q', p)$, and a CRWL proof proving $\mathcal{P} \vdash (\texttt{sxq Q2})\theta_2 \rightarrow \texttt{t}$.

  Observe that we are assuming that each query introduces new variable names. Then we can define $\theta = \theta_1 \cup \theta_2$ without ambiguity. Now, applying a variant of the seventh rule of `sxq` (for instance `sxq (xif (cond Q1') Q2') = sxq Q2' <== sxq Q1' == A` , see Figure 5), and defining a substitution $\sigma = \{Q'_1 \mapsto Q_1, Q'_2 \mapsto Q_2, A \mapsto t'\} \cdot \theta$ (in fact A can be bound to any $t' \in L''$), we can build a CRWL proof for $\mathcal{P} \vdash (\texttt{sxq Q})\theta \rightarrow \texttt{t}$:

$$\frac{\begin{array}{l}(\texttt{xif (cond Q1) Q2})\theta \rightarrow (\texttt{xif (cond Q1') Q2'})\sigma \\ (\texttt{sxq Q1'})\sigma == A\sigma \\ (\texttt{sxq Q2'})\sigma \rightarrow \texttt{t}\end{array}}{(\texttt{sxq xif cond Q1 Q2})\theta \rightarrow \texttt{ t}}$$

Taking into account the definition of σ the previous (FA) step can be rewritten as:

$$\frac{\begin{array}{l}(\texttt{xif (cond Q1) Q2})\theta \rightarrow (\texttt{xif (cond Q1) Q2})\theta \\ \texttt{sxq Q1}\theta_1 == \texttt{t}' \\ \texttt{sxq Q2}\theta_2 \rightarrow \texttt{t}\end{array}}{(\texttt{sxq xif (cond Q1) Q2 })\theta \rightarrow \texttt{ t}}$$

In the first premise we have the same term at left-hand side and at right-hand side, and the existence of the proof is ensured by Lemma 4. The same Lemma indicates that proving $\texttt{sxq Q1}\theta_1 == \texttt{t}'$ is equivalent to proving $\texttt{sxq Q1}\theta_1 \rightarrow \texttt{t}'$, and we have seen that it holds by induction hypothesis. The same happens with the third premise.

– $Q \equiv$ `if` $(\$x_i := \$x_j)$`then` Q_1.
In $\mathcal{TOY}$: $\texttt{xif (xp (var X}_i\texttt{) := xp (var X}_j\texttt{)) Q1}$.

In this case, by $\mathbf{XQ_6}$,

$$\begin{array}{lll}[\![\text{if } (\$x_i := \$x_j) \text{ then } Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k) := & & \\ \qquad [\![Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k) & & \text{if } t_i = t_j \\ \qquad (\mathcal{F}, [\;]) & & \text{e.o.c}\end{array}$$

If $t_i \neq t_j$ the result holds. Thus we assume that $t_i = t_j$, and that:

$$[\![Q_1]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}', L')$$

Let t be any element of L'. We must prove that $\mathcal{P} \vdash \texttt{sxq Q}\theta \rightarrow \texttt{t}$ for some substitution θ such that $\theta(\$x_i) = t_i$ for every $\$x_i \in Rel(Q', p)$. In first place it is possible to check that $\mathcal{P} \vdash (\texttt{sxq (xp (var X}_i\texttt{)) == sxq (xp (var X}_j\texttt{)))}\theta$ with θ such that $\theta(\$x_i) = t_i$, $\theta(\$x_j) = t_j$.

$$\frac{\dfrac{\dfrac{(\texttt{X}_i)\theta \rightarrow \texttt{t}'}{(\texttt{sxqPath (var X}_i\theta)) \rightarrow \texttt{t}'}}{\texttt{sxq (xp (var X}_i\theta)) \rightarrow \texttt{t}'} \qquad \dfrac{\dfrac{(\texttt{X}_j)\theta \rightarrow \texttt{t}'}{(\texttt{sxqPath (var X}_j\theta)) \rightarrow \texttt{t}'}}{(\texttt{sxq (xp (var X}_j\theta)) \rightarrow \texttt{t}'}}{\texttt{sxq (xp (var X}_i\theta)) == \texttt{sxq (xp (var X}_j\theta))}$$

With $t' \equiv t_i \equiv t_j$, using the inference rules (JN), (FA) using the first rule of `sxq`, (FA) using the first rule of `sxqPath` and finally proving the premises on top applying the Lemma 4.
By Lemma 3, $Rel(Q', p \cdot 1) = Rel(Q', p)$, and the induction hypothesis can be applied to Q_1 as in the previous case. Now, applying the sixth rule of `sxq` (`sxq (xif (Q2 := Q3) Q1) = sxq Q1 <== sxq Q2 == sxq Q3`, see Figure 5), it is possible to find a CRWL proof for $\mathcal{P} \vdash (\texttt{sxq Q})\theta \rightarrow \texttt{t}$ (the details are similar to the previous cases).

– $Q \equiv \$x_i$.
In this case, $[\![\$x_i]\!]_k(\mathcal{F}, t_1, \ldots, t_k) := (\mathcal{F}, [t_i])$ by $\mathbf{XQ_3}$. The query Q is such that $Q'_{|p} \equiv Q$ for $p \in Pos(Q')$. Then, $\$x_i \in Rel(Q', p) = \{\$x_1, \ldots, \$x_k\}$.

Then $\$x_i \in Rel(Q,p)$. θ must be a substitution such that $(\$x_i)\theta = t_i$. The representation in $\mathcal{TOY}$ of $\$x_i$ is `sxqPath (var Xi)`, see Figure 5), and applying the first rule of `sxqPath`, (`sxqPath (var X) = X`, see Figure 5), we can build a CRWL proof for $\mathcal{P} \vdash (\texttt{sxq Q})\theta \to t_i$ with instance $\sigma = \{X \to t_i\}$.

$$\frac{(\texttt{varX}_i\theta) \to (\texttt{varX}\sigma) \qquad (\texttt{X}_i)\sigma \to\ t_i}{\texttt{sxqPath}\ (\texttt{varX}_i)\theta \to\ t_i}$$

Applying the definition of θ and σ in the premises we have:

$$\frac{(\texttt{varX}_i\theta) \to (\texttt{varX}\sigma) \qquad (\texttt{X}_i)\sigma \to\ t_i}{\texttt{sxqPath}\ (\texttt{varX}_i)\theta \to\ t_i}$$

and all the premises are consequence of Lemma 4

– $Q \equiv \$x_i/axis :: \nu$. The proof in this case is very similar to the corresponding case in Lemma 5 which can be in fact read as an *if and only if* proof.

As in the case of correctness, the completeness theorem is just a particular case of the Lemma:

Theorem 2. *Let P be the $\mathcal{TOY}$ program of Figure 5. Let Q be a SXQ query with $free(Q) = \{\$x_1, \ldots, \$x_k\}$ and suppose that $[\![Q]\!]_k(\mathcal{F}, t_1, \ldots, t_k) :=^* (\mathcal{F}', L)$ for some $\mathcal{F}$, $\mathcal{F}', t_1, \ldots, t_k, L$. Then, for every $t \in L$, there is a substitution θ such that $\theta(\$x_i) = t_i$ for all $\$x_i \in free(Q)$ and a CRWL-proof proving $\mathcal{P} \vdash sxq\ Q\theta ==$ `t`.*

Proof. In Lemma 6, consider $p \equiv \varepsilon$ and thus $Q' \equiv Q$. Then $Rel(Q,\varepsilon) = free(Q) \cup V_{for}(Q,\varepsilon) = free(Q)$. Then the conclusion of the lemma is the same as the conclusion of the Theorem.

5 Application: Test Case Generation

In this chapter we show how the embedding of SXQ in $\mathcal{TOY}$ can be used for obtaining test-cases for the queries. For instance, consider the erroneous query of the next example.

Example 8. Suppose that the user also wants to include the publisher of the book among the data obtained in Example 1. The following query tries to obtain this information:

```
Q = for $b in doc("bib.xml")/bib/book,
        $r in doc("reviews.xml")/reviews/entry,
        where $b/title = $r/title
        for $booktitle in $r/title,
            $revtext in $r/review,
            $publisher in $r/publisher
        return  <rev> $booktitle $publisher $revtext </rev>
```

However, there is an error in this query, because in the expression `$r/publisher` the variable `$r` should be `$b`, since the publisher is in the document "bib.xml", not in "reviews.xml". The user does not notice that there is an error, tries the query (in $\mathcal{TOY}$ or in any XQuery interpreter) and receives an empty answer.

In order to check whether a query is erroneous, or even to help finding the error, it is sometimes useful to have test-cases, i.e., XML files which can produce some answer for the query. Then the test-cases and the original XML documents can be compared, and this can help finding the error. In our setting, such test-cases are obtained for free, thanks to the generate and test capabilities of logic programming. The general process can be described as follows:

1. Let `Q'` be the translation of the SXQ query Q as a $\mathcal{TOY}$ dataterm by means the primitive `parse_xquery`.
2. Let $F_1, \ldots, F_k$ be the names of the XML documents occurring in `Q'`.
3. Let `Q"` be the result of replacing each expression of the form `doc(`F_i`)` by a new variable D_i, for $i = 1 \ldots k$.
4. Let `"expected.xml"` be a document containing an expected answer for the query `Q`.
5. Try the following goal:

```
Toy> sxq Q'' == load_doc "expected.xml",
                write_xml_file D1 F1',
                ... ,
                write_xml_file Dk Fk'
```

The idea is that the goal above looks for values of the logic variables D_i fulfilling the strict equality. The result is that after solving this goal, the D_i variables contain XML documents that can produce the expected answer for this query. Then each document is saved into a new file with name F_i'. For instance F_i' can consist of the original name F_i preceded by some suitable prefix *tc*. The process can be automatized, and the result is the code of Figure 6.

The code uses the list concatenation operator `++` which is defined in $\mathcal{TOY}$ as usual in functional languages such as Haskell. It is worth observing that if there are no test-case documents that can produce the expected result for the query, the call to `generateTC` will loop. The next example shows the generation of test-cases for the wrong query of Example 8.

Example 9. Consider the query of Example 8 , and suppose the user writes the following document "expected.xml":

```
<rev>
 <title>Some title</title>
 <review>The review</review>
 <publisher>Publisher</publisher>
</rev>
```

This is a possible expected answer for the query. Now we can try the goal:

```
prepareTC (xp E)            = (xp E',L)
                              where (E',L) = prepareTCPath E
prepareTC (xmlExp X)        = (xmlExp X, [])
prepareTC (comp Q1 Q2)      = (comp Q1' Q2', L1++L2)
                               where (Q1',L1) = prepareTC Q1
                                     (Q2',L2) = prepareTC Q2
prepareTC (xfor X Q1 Q2)    = (xfor X Q1' Q2', L1++L2)
                              where (Q1',L1) = prepareTC Q1
                                    (Q2',L2) = prepareTC Q2
prepareTC (xif (Q1:=Q2) Q3) = (xif (Q1':=Q2') Q3',L1++(L2++L3))
                               where (Q1',L1) = prepareTC Q1
                                     (Q2',L2) = prepareTC Q2
                                     (Q3',L3) = prepareTC Q3
prepareTC (xif (cond Q1) Q2) = (xif (cond Q1) Q2, L1++L2)
                               where (Q1',L1) = prepareTC Q1
                                    (Q2',L2) = prepareTC Q2

prepareTCPath (var X)   = (var X, [])
prepareTCPath (X :/ S)  = (X :/ S, [])
prepareTCPath (doc F S) = (A :/ S, [write_xml_file A ("tc"++F)])

generateTC Q F = true <==  sxq Qtc == load_doc F, L==_
               where (Qtc,L) = prepareTC Q
```

Fig. 6. $\mathcal{TOY}$ test case generation rules for SXQ

```
Toy> Q == parse_xquery "for....", R == generateTC Q "expected.xml"
```

The first strict equality parses the query, and the second one generates the XML documents which constitute the test cases. In this example the test-cases obtained are:

```
% bibtc.xml
<bib>
 <book>
  <title>Some title</title>
 </book>
</bib>
```

```
% revtc.xml
<reviews>
 <entry>
  <title>Some title</title>
  <review>The review </review>
  <publisher>Publisher</publisher>
 </entry>
</reviews>
```

By comparing the test-case “revtc.xml" with the file “reviews.xml" (see Appendix A) we observe that the publisher is not part of the structure defined for reviews. Then, it is easy to check that in the query the publisher is obtained from the reviews instead of from the *bib* document, and that this constitutes the error.

6 Conclusions

The report shows the embedding of a fragment of the XQuery language for querying XML documents in the functional-logic language $\mathcal{TOY}$. Although only a small subset of XQuery consisting only of *for*, *where/if* and *return* statements has been considered, the users of $\mathcal{TOY}$ can now perform simple queries typical of database queries such as *join* operations. The embedding has respected the declarative nature of $\mathcal{TOY}$, and we have provided the soundness of the approach with respect to the operational semantics of XQuery. From the point of view of XQuery the results are also encouraging. The embedding allows the user to generate test-cases automatically when possible, which is useful for testing the query, or even for helping to find the error in the query.

The most obvious future work would be introducing *let* statements, which presents two novelties. The first is that they are *lazy*, that is, they are not evaluated if they are not required by the result. This part is easy to fulfill since we are in a lazy language. In particular, they could be introduced as local definitions (*where* statements in $\mathcal{TOY}$).

The second novelty is more difficult to capture, and it is that the variables introduced by *let* represent an XML sequence. The natural representation in $\mathcal{TOY}$ would be a list, but the non-deterministic nature of our proposal does not allow us to collect all the results provided by an expression in a declarative way. A possible idea would be to use the functional-logic Curry [10] and its encapsulated-search [12], or even the non-declarative *collect* primitive included in $\mathcal{TOY}$. In any case, this will imply a different theoretical framework and new proofs for the results. A different line for future work is the use of test cases for finding the error in the query using some variation of declarative debugging [18] that would be applied to this setting.

References

1. J. Almendros-Jiménez. An Encoding of XQuery in Prolog. In Z. Bellahsène, E. Hunt, M. Rys, and R. Unland, editors, *Database and XML Technologies*, volume 5679 of *Lecture Notes in Computer Science*, pages 145–155. Springer Berlin / Heidelberg, 2009.
2. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1999.
3. M. Benedikt and C. Koch. From XQuery to relational logics. *ACM Trans. Database Syst.*, 34:25:1–25:48, December 2009.
4. R. Caballero, Y. García-Ruiz, and F. Sáenz-Pérez. Integrating XPath with the Functional-Logic Language $\mathcal{TOY}$. In *Proceedings of the 13th international conference on Practical aspects of declarative languages*, volume 6539 of *PADL'11*, pages 145–159, Berlin, Heidelberg, 2011. Springer-Verlag.
5. D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0: An XML Query Language, 2010. `http://www.w3.org/TR/xquery/`.
6. J. Clark. XML Path Language (XPath) 2.0, 2010. `http://www.w3.org/TR/xpath20/`.

7. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '77, pages 238–252, New York, NY, USA, 1977. ACM.
8. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992. (The editor of Journal of Logic Programming has mistakenly published the unreadable galley proof. For a correct version of this paper, see `http://www.di.ens.fr/~cousot`.).
9. L. Fegaras. Propagating updates through XML views using lineage tracing. In *IEEE 26th International Conference on Data Engineering (ICDE)*, pages 309 – 320, march 2010.
10. M. Hanus. Curry: An Integrated Functional Logic Language (version 0.8.2 march 28, 2006). Available at: `http://www.informatik.uni-kiel.de/~mh/curry/`, 2003.
11. M. Hanus. Declarative processing of semistructured web data. In J. Gallagher and M. Gelfond, editors, *Technical Communications of the 27th International Conference on Logic Programming (ICLP'11)*, volume 11 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 198–208, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
12. M. Hanus and F. Steiner. Controlling search in declarative programs. In *Principles of Declarative Programming (Proc. Joint International Symposium PLILP/ALPŠ98*, pages 374–390. Springer LNCS, 1998.
13. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial evaluation and automatic program generation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
14. F. J. López-Fraguas and J. S. Hernández. $\mathcal{TOY}$: A Multiparadigm Declarative System. In *RTA '99: Proceedings of the 10th International Conference on Rewriting Techniques and Applications*, pages 244–247, London, UK, 1999. Springer-Verlag.
15. J. C. G. Moreno, M. T. Hortalá-González, F. J. López-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. *J. Log. Program.*, 40(1):47–87, 1999.
16. D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking Forward. In *Workshop on XML-Based Data Management*, pages 109–127. Springer, 2002.
17. D. Seipel, J. Baumeister, and M. Hopfner. Declaratively Querying and Visualizing Knowledge Bases in XML. In *In Proc. Int. Conf. on Applications of Declarative Programming and Knowledge Management*, pages 140–151. Springer, 2004.
18. E. Y. Shapiro. *Algorithmic Program Debugging*. ACM Distiguished Dissertation. MIT Press, 1982.
19. W3C. Extensible Markup Language (XML), 2007.
20. W3C. XQuery 1.0 and XPath 2.0 Formal Semantics (Second Edition), 2010. `http://www.w3.org/TR/xquery-semantics/`.
21. P. Walmsley. *XQuery*. O'Reilly Media, Inc., 2007.

A Examples of XML documents

```
% bib.xml

<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>

    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>

    <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>39.95</price>
    </book>

    <book year="1999">
        <title>The Economics of Technology and Content for Digital TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>

</bib>
```

```
% reviews.xml

<reviews>
    <entry>
        <title>Data on the Web</title>
        <price>34.95</price>
        <review>
               A very good discussion of semi-structured database
               systems and XML.
        </review>
    </entry>
    <entry>
        <title>Advanced Programming in the Unix environment</title>
        <price>65.95</price>
        <review>
               A clear and detailed discussion of UNIX programming.
        </review>
    </entry>
    <entry>
        <title>TCP/IP Illustrated</title>
        <price>65.95</price>
        <review>
               One of the best books on TCP/IP.
        </review>
    </entry>
</reviews>
```