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, ygarcia@fdi.ucm.es

April 2011
(Revised April 18, 2011)

* 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 paper addresses the problem of integrating a fragment of XQuery, a language for querying XML documents, into the functional-logic language 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 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 and functional programming. In this paper we consider the introduction of a simple subset of XQuery into the functional-logic language TOY.

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, partial evaluation or algorithmic debugging. 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 [9], where rule-based language for processing semistructured data that is implemented and embedded in the functional logic language Curry, and also in [13] 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 TOY. This is the approach considered in this paper.

Thus, our goal is to include XQuery using the purely declarative features of the languages. This allows us to prove that the semantics of the fragment of XQuery has been correctly included in 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 [5], where XPath expressions were introduced in 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 paper with respect to [5] 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 is correctly represented in TOY.

Next Section introduces the fragment of XQuery considered and a suitable operational semantics for evaluating queries. Then the language TOY and its semantics are presented in Section 3. Section 4 includes the interpreter that
performing the evaluation of simple XQuery expressions in $T_OY$. The theoretical results establishing the soundness of the approach with respect to the operational semantics of Section 2 are presented in Section 4.1. Section 5 explains the automatic generation of Test Cases for simple XQuery expressions. Finally, Section 6 concludes summarizing the results and proposing future work.

An extended version of the paper including proofs of the theoretical results can be found at [2].

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 [18,20]. The syntax and semantics of the language are quite complex [19], and thus only a small subset of the language is usually considered. The next subsection introduces the fragment of XQuery considered in this paper.

2.1 The subset SXQ

In [4] 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 paper we consider a simplified version of XQ, which we call SXQ and whose syntax can be found in Figure 1. 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 only admits either variables or other tags inside a tag.

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 $lets$ 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.

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

\[
\text{query ::= ( ) | query query | tag} \\
\text{ | var | var/axis ::= $\nu$} \\
\text{ | for var in query return query} \\
\text{ | if cond then query} \\
\text{cond ::= var = var | query} \\
\text{tag ::= $\langle a \rangle var/\langle /a \rangle$ | $\langle a \rangle tag/\langle /a \rangle$}
\]
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 [17], sample data 1.1.2 and 1.1.4 to check the structure of these documents and an example). 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 the different books, and $r the different reviews. The *where* condition ensures that only reviews corresponding to the book are considered. Finally, the last two 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 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 variable and each one consisting of a single step.

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

```
for $x1 in doc("bib.xml")/child::bib return
    for $x2 in $x1/child::book return
        for $x3 in doc("reviews.xml")/child::reviews return
            for $x4 in $x3/entry return
                if ($x2/title= $x4/title) then
                    for $x5 in $x4/title return
                        for $x6 in $x4/review return <rev> $x5 $x6 </rev>
```

We end this subsection with a few definitions that are useful for the rest of the paper. The set of variables in a query $Q$ is represented as $Var(Q)$. Given a query $Q$, we use the notation $Q_p$ for representing the subquery $Q'$ that can be found in $Q$ at position $p$. Positions are defined as usual in syntax trees:
Lemma 1. Given a query $Q$ and a position $p$, $Q_{|p}$ is defined as follows:

\[
\begin{align*}
Q_{|p} &= Q, \\
(Q_1, Q_2)_{|(i,p)} &= (Q_i)_{|p}, \quad i \in \{1, 2\} \\
(&\text{for var in } Q_1 \text{ return } Q_2)_{|(i,p)} &= (Q_i)_{|p}, \quad i \in \{1, 2\} \\
(&\text{if } Q_1 \text{ then } Q_2)_{|(i,p)} &= (Q_i)_{|p}, \quad i \in \{1, 2\} \\
(&\text{if var = var then } Q_1)_{|(1,p)} &= (Q_1)_{|p}
\end{align*}
\]

Hence the position of a subquery is the path in syntax the tree represented as concatenation of children positions $p_1 \cdot p_2 \ldots \cdot p_n$. For every position $p$, $\varepsilon \cdot p = p \cdot \varepsilon = p$. In general $Q_p$ is not a proper SXQ query, since it can contain free variables, which are variables defined previously in for statements in $Q$. The set of variables of $Q$ that are relevant for $Q_p$ is the subset of $\text{Var}(Q)$ that can appear free in any subquery at position $p$. This set, denoted as $\text{Rel}(Q, p)$ is defined recursively as follows:

Definition 2. Given a query $Q$, and a position $p$, $\text{Rel}(Q, p)$ is defined as:

1. $\emptyset$, if $p = \varepsilon$.
2. $\text{Rel}(Q', p')$, if $Q \equiv \langle a \rangle Q'/a$, and $p = 1 \cdot p'$.
3. $\text{Rel}(Q_1, p')$, if $Q \equiv Q_1 Q_2, p = 1 \cdot p'$.
4. $\text{Rel}(Q_2, p')$, if $Q \equiv Q_1 Q_2, p = 2 \cdot p'$.
5. $\text{Rel}(Q_1, p')$, if $Q \equiv \text{for } X_i \text{ in } Q_1 \text{ return } Q_2, p = 1 \cdot p'$.
6. $\{X_i\} \cup \text{Rel}(Q_2, p')$, if $Q \equiv \text{for } X_i \text{ in } Q_1 \text{ return } Q_2, p = 2 \cdot p'$.
7. $\text{Rel}(Q_1, p')$, if $Q \equiv \text{if } Q_1 \text{ then } Q_2, p = 1 \cdot p'$.
8. $\text{Rel}(Q_2, p')$, if $Q \equiv \text{if } Q_1 \text{ then } Q_2, p = 2 \cdot p'$.

Observe that the cases $Q \equiv ()$, $Q \equiv X_i$, $Q \equiv X_i/\chi :: \nu$, and $X_i = X_j$ corresponds to $p \equiv \varepsilon$.

Without loss of generality we assume that all the relevant variables for a given position are indexed starting by 1 from the outer level. We also assume that every for statement introduces a new variable. A query like $\text{for } X \text{ in } ((\text{for } Y \text{ in } \ldots) (\text{for } Y \text{ in } \ldots)) \ldots$ is then renamed to an equivalent query of the form $\text{for } X_i \text{ in } ((\text{for } Y_2 \text{ in } \ldots) (\text{for } Z_2 \text{ in } \ldots)) \ldots$ (notice that the two $Y$ variables occurred in different scopes).

The next lemma introduces a basic property of $\text{Rel}(Q, p)$.

Lemma 1. Let $Q$ be a SXQ query, and $(p_1 \cdot \ldots \cdot p_n)$ a valid position in $Q$. Then, for every $k$, $1 \leq k \leq n$, $\text{Rel}(Q, (p_1 \cdot \ldots \cdot p_k)) \cup \text{Rel}(Q, (p_1 \cdot \ldots \cdot p_k)) = \text{Rel}(Q, (p_1 \cdot \ldots \cdot p_k)) \cup \text{Rel}(Q, (p_{k+1} \cdot \ldots \cdot p_n))$.

Proof. By complete induction on $k$.

– Base case: $k = 1$. The result: $\text{Rel}(Q, (p_1 \cdot \ldots \cdot p_m)) = \text{Rel}(Q, p_1) \cup \text{Rel}(Q, (p_2 \cdot \ldots \cdot p_m))$ can be proven applying induction on the structure of $Q$:
• $Q = Q_1 Q_2$. In this case, $p_1$ can be either 1 or 2. Suppose $p_1 = 1$. Then:
  (1) $Q_{11} = Q_1$ (by definition 1)
  (2) $\text{Rel}(Q, 1) = \text{Rel}(Q_1, \varepsilon) = \emptyset$ (by definition 2, rule 3)

By Definition 2, rule 3, $\text{Rel}(Q, Q_2, (1 \cdot p_2 \cdot \ldots \cdot p_m)) = \text{Rel}(Q_1, (p_2 \cdot \ldots \cdot p_m))$.

Then $\text{Rel}(Q_1, (p_2 \cdot \ldots \cdot p_m)) = \emptyset \cup \text{Rel}(Q_1, (p_2 \cdot \ldots \cdot p_m))$ and by (1) and (2), we have $\text{Rel}(Q_1, (p_2 \cdot \ldots \cdot p_m)) = \text{Rel}(Q, 1) \cup \text{Rel}(Q_1, (p_2 \cdot \ldots \cdot p_m))$ and the result holds. If $p_1 = 2$, the result can be proved similarly.

• $Q \equiv X_i$ in $Q_1$ return $Q_2$. If $p_1 = 1$ the result can be proved similarly to the previous case, because the set of relevant variables does not change. Now, suppose $p_1 = 2$. Then:
  (1) $Q_{12} = Q_2$ (by definition 1)
  (2) $\text{Rel}(Q, 2) = \{X_i\} \cup \text{Rel}(Q_2, \varepsilon) = \{X_i\}$ (by definition 2, rule 6)

By Definition 2, rule 6 and by (2) and (1), $\text{Rel}(Q_1, (p_2 \cdot \ldots \cdot p_m)) = \text{Rel}(Q_2, (p_2 \cdot \ldots \cdot p_m)) = \text{Rel}(Q_1, (p_2 \cdot \ldots \cdot p_m)) = \text{Rel}(Q, 2) \cup \text{Rel}(Q_2, (p_2 \cdot \ldots \cdot p_m))$, which proves the result.

The rest of the base cases can be proved similarly.

Inductive step: Assume the result holds for $k = 1 \ldots n$. In particular for $k = n$:

(i) $\text{Rel}(Q, (p_1 \cdot \ldots \cdot p_m)) = \text{Rel}(Q, (p_1 \cdot \ldots \cdot p_n)) \cup \text{Rel}(Q_1, (p_{n+1} \cdot \ldots \cdot p_m))$

And for $k = 1$ applied to $\text{Rel}(Q, (p_1 \cdot \ldots \cdot p_n), (p_{n+1} \cdot \ldots \cdot p_m))$:

(ii) $\text{Rel}(Q, (p_1 \cdot \ldots \cdot p_n), (p_{n+1} \cdot \ldots \cdot p_m)) = \text{Rel}(Q, (p_1 \cdot \ldots \cdot p_n), (p_{n+1} \cdot \ldots \cdot p_m)) \cup \text{Rel}(Q_1, (p_{n+1} \cdot \ldots \cdot p_m)) = \text{Rel}(Q, (p_1 \cdot \ldots \cdot p_n), (p_{n+1} \cdot \ldots \cdot p_m)) \cup \text{Rel}(Q_1, (p_{n+1} \cdot \ldots \cdot p_m))$

Then, by (i) and (ii):

$\text{Rel}(Q, (p_1 \cdot \ldots \cdot p_m)) = \text{Rel}(Q, (p_1 \cdot \ldots \cdot p_n)) \cup \text{Rel}(Q, (p_{n+1} \cdot \ldots \cdot p_m)) = \text{Rel}(Q, (p_1 \cdot \ldots \cdot p_n)) \cup \text{Rel}(Q, (p_{n+1} \cdot \ldots \cdot p_n)) \cup \text{Rel}(Q, (p_{n+2} \cdot \ldots \cdot p_m))$

which completes the proof.

2.2 XQ Operational Semantics

Figure 2 introduces the operational semantics of XQ that can be found in [4]. The only difference with respect to the semantics of this paper is that there is no rule for the constructor $lab$, for the sake of simplicity.

As explained in [4], the operator $\text{construct}(\alpha, (F, [w_1 \ldots w_n]))$, denotes construction of a new tree, where $\alpha$ is a label, $F$ is a data forest, and $[w_1 \ldots w_n]$ is a...
The symbol $\text{child of root}($ returns an indexed forest ($F$ is a list of nodes in $\mathcal{F}$) being an isomorphic copy of the subtree rooted by $w_1$ in $\mathcal{F}$. The symbol $\bigcup$ used in the rules takes two indexed forests $(\mathcal{F}_1,l_1),(\mathcal{F}_2,l_2)$ and returns an indexed forest $(\mathcal{F}_1 \cup \mathcal{F}_2,l)$, where $l$ is the concatenation of $l_1$ and $l_2$. Without loss of generality this semantics assumes that all the variables relevant for a subquery are numbered consecutively starting by 1 as in Example 2. It also assumes that the documents appear explicitly in the query. That is, in Example 2 we must suppose that instead doc(“bib.doc”) we have the XML corresponding to this document. Of course this is not feasible in practice, but simplifies the theoretical setting and it is assumed in the rest of the paper. These semantic rules constitute a term rewriting system (TRS in short, see [3]), with each rule defining a single reduction step. The symbol $\Rightarrow^*$ represents the reflexive and transitive closure of $\Rightarrow$ as usual. The TRS is terminating and confluent (the rules are not overlapping). Normal forms have the shape $(\mathcal{F},t_1,\ldots,t_n)$ where $\mathcal{F}$ is a forest of XML fragments, and $t_i$ nodes in $\mathcal{F}$, meaning that the query returns the XML fragments $t_1,\ldots,t_n$. The semantics evaluates a query starting with the expression $[Q_0]_n(\emptyset,())$. Along intermediate steps, expressions of the form $[Q']_n(\mathcal{F},\tau_k)$ are obtained. The idea is that $Q'$ is a subquery of $Q$ with $k$ relevant variables (which can occur free in $Q'$), that must take the values $\tau_k$. The next lemma formalizes these ideas.

**Lemma 2.** Let $Q$ be a SXQ query. Suppose that

$$[Q]_0(\emptyset,()) := \Rightarrow^* [Q']_n(\mathcal{F},\tau_n)$$

Then:
- $Q'$ is a subquery of $Q$, that is, $Q' = Q_p$ for some $p$.
- $\text{Rel}(Q_p) = \{X_1,\ldots,X_n\}$.
- Let $S$ be the set of free variables in $Q'$. Then $S \subset \text{Rel}(Q,p)$.

---

**Fig. 2.** Semantics of Core XQuery
\[ [Q']_n(F, \tau_n) = [Q']_0(\emptyset, ()), \text{ with } \theta = \{X_1 \mapsto e_1, \ldots, X_n \mapsto e_n\} \]

Proof. Straightforward from Definition 2 and from the XQ semantic rules of Figure 2.

A more detailed discussion about this semantics and its properties can be found in [4].

3 $\texttt{TOY}$ and Its Semantics

A $\texttt{TOY}$ [11] program is composed of data type declarations, type alias, infix operators, function type declarations and defining rules for function symbols. The syntax of partial expressions in $\texttt{TOY}$ $e \in \text{Exp}_\bot$ is $e ::= \bot \mid X \mid h \{e'\}$ where $X$ is a variable and $h$ either a function symbol or a data constructor. Expressions of the form $\{e'\}$ stand for the application of expression $e$ (acting as a function) to expression $e'$ (acting as an argument). Similarly, the syntax of partial patterns $t \in \text{Pat}_\bot \subset \text{Exp}_\bot$ can be defined as $t ::= \bot \mid X \mid c t_1 \ldots t_m \mid f t_1 \ldots t_m$ 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 partial patterns for all $1 \leq i \leq m$.

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

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

The data type $\texttt{node}$ 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 $\texttt{tag}$ includes the tag name (an argument of type $\texttt{string}$) followed by a list of attributes, and finally a list of child nodes. The data type $\texttt{attribute}$ contains the name of the attribute and its value (both of type $\texttt{string}$). The last type alias, $\texttt{xml}$, renames the data type $\texttt{node}$. 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 $\texttt{TOY}$ has the form:

\[
\left( \frac{f t_1 \ldots t_n \rightarrow r}{e_1, \ldots, e_k \Leftarrow s_1 = u_1, \ldots, s_m = u_m} \right)
\]

where $u_i$ and $r$ are expressions (that can contain new extra variables) and $t_i$, $s_i$ are patterns.

In $\texttt{TOY}$, variable names must start with either an uppercase letter or an underscore (for anonymous variables), whereas other identifiers start with lowercase.
TOY includes two primitives for loading and saving XML documents, called `load_xml_file` and `write_xml_file` respectively. For convenience all the documents are started with a dummy node `root`. 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) [7] 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), the (JN) (join) rule that indicates how to prove strict equalities, and the function application rule (FA). In every inference rule, `e, e_i ∈ Exp_⊥` are partial expressions

**BT**

```
e → ⊥
```

**RR**

```
X → X with X ∈ Var
```

**DC**

```
e_1 → t_1 ... e_m → t_m

h τ_m → h t_m
```

```
h τ_m ∈ Pat_⊥
```

**JN**

```
e → t e' → t

e == e'
```

```
t ∈ Pat (total pattern)
```

**FA**

```
e_1 → t_1 ... e_n → t_n C r π_k → t

f τ_n π_k → t
```

```
if (f τ_n → r ≡ C) ∈ [P]_⊥, t ≠ ⊥
```

**Fig. 3.** CRWL Semantic Calculus

and `t_i, t, s ∈ Pat_⊥` are partial patterns. The notation `[P]_⊥` of the inference rule `FA` represents the set `{(l → r ≡ C)θ | (l → r ≡ C) ∈ P, θ ∈ Subst_⊥}` of partial instances of the rules in 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 τ_n`:

1. Obtain partial patterns `t_i` as suitable approximations of the arguments `e_i`.
2. Apply a program rule `(f τ_n → r ≡ C) ∈ [P]_⊥`, 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 \texttt{where} are 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 \texttt{member} that returns all the elements in a list:

\[
\text{member} :: [A] \rightarrow A \\
\text{member} [X | Xs] = X \\
\text{member} [X | Xs] = \text{member} Xs
\]

Another example of $\mathcal{TOY}$ function is the definition of the infix operator $\ldots$:

\[
\text{infixr 90 .::. (\ldots :: (A \rightarrow B) \rightarrow (B \rightarrow C) \rightarrow (A \rightarrow C)} \\
(F .::. G) X = G (F X)
\]

As the examples show, $\mathcal{TOY}$ is a typed language. However the type declaration is optional and in the rest of the paper 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 $\text{JN}$) if both $e_1$ and $e_2$ can be reduced to the same total pattern $t$. For instance, the goal $\text{member} [1,2,3,4] == R$ yields four answers, the four values for $R$ that make the equality true: $\{R \rightarrow 1\}, \ldots, \{R \rightarrow 4\}$.

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

**Lemma 3.** Let $t_1,t_2$ be patterns and $e$ be an expression. Then

1. If $P \vdash t_1 \rightarrow t_2$, and $t_2$ is total, then $t_1 \equiv t_2$ (the symbol $\equiv$ is used to represent syntactic equivalence).
2. If $P \vdash t_1 == t_2$, then $t_1$ and $t_2$ are syntactically equal total patterns.
3. $P \vdash e == t_1$, iff $P \vdash e \rightarrow t_1$ and $t_1$ total.
4. It is always possible to prove $P \vdash t_1 \rightarrow t_1$.

**Proof.**

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

In order to represent SXQ queries in \( \mathcal{T}O\mathcal{Y} \) 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 \( \text{sxq} \) allows representing any SXQ query (see SXQ syntax in Figure 1). It is worth noticing that a variable introduced by a \textit{for} statement has type \( \text{xml} \), indicating that the variable always contains a value of this type. \( \mathcal{T}O\mathcal{Y} \) includes a primitive \texttt{parse\_xquery} that translates any SXQ expression into its corresponding representation as a term of this datatype, as the next example shows:

\textit{Example 3.} The translation of the SXQ query of Example 2 into the datatype \( \text{sxq} \) produces the following \( \mathcal{T}O\mathcal{Y} \) data term:

```
Toy> parse\_xquery "for $x1 in doc("bib.xml")/child::bib return
  for $x2 in ..... <rev> $x5 $x6 </rev>" == R
yes
{R --> xfor X1 (xp (doc "bib.xml" (child ::. (nameT "bib"))))
  (xfor X2 (xp ( X1 ::/ (child ::. (nameT "book")))))
  (xfor X3 (xp (doc "reviews.xml" (child ::. (nameT "reviews")))))
  (xfor X4 (xp ( X3 ::/ (child ::. (nameT "entry")))))
  (xif (((xp(X2 ::/ (child ::. (nameT "title")))) :=
      (xp(X4 ::/ (child ::. (nameT "title"))))))
  (xfor X5 (xp ( X4 ::/ (child ::. (nameT "title")))))
  (xfor X6 (xp ( X4 ::/ (child ::. (nameT "review"))))
    (xmlExp (xmlTag "rev" [] [X5,X6]))))}
```

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 Section 3.

The rules of the \( \mathcal{T}O\mathcal{Y} \) interpreter that processes SXQ queries can be found in Figure 4. The main function is \( \text{sxq} \), which distinguishes cases depending of the form of the query. If it is an XPath expression then the auxiliary function \( \text{sxqPath} \) 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 (\( \text{comp} \) construct), the result of evaluating any of them is returned using non-determinism. The \textit{for} statement (\( \text{xfor} \) construct) forces the evaluation of the query \( Q_1 \) and binds the variable \( X \) to the result. Then the result query \( Q_2 \) is evaluated. The case of the \textit{if} statement is analogous. The XPath subset considered includes tests for attributes (\texttt{attr}), label names (\texttt{nameT}), general elements (\texttt{nodeT}), text nodes (\texttt{textT}) and comments.
Fig. 4. $TOY$ transformation rules for SXQ

...
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 paper 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 \( \{X_1, \ldots, X_n\} \) we use the notation \( \theta(V) \) to indicate the sequence \( \theta(X_1), \ldots, \theta(X_n) \). In these results it is implicitly assumed that there is a bijective mapping \( f \) from XML format to the datatype \( \text{xml} \) in \( T\omega\). Also, variables in XQuery $\$x_i$ are assumed to be represented in \( T\omega \) as \( X_i \) and conversely. However, in order to simplify the presentation, we omit the explicit mention to \( f \) and to \( f^{-1} \).

**Lemma 4.**

Let \( P \) be a \( T\omega \) program, \( Q' \) an SXQ query, and \( Q, p \) such that \( Q \equiv Q'_p \). Define \( V = \text{Rel}(Q'_p, p) \) (see Definition 3), and \( k = |V| \). Let \( \theta \) be a substitution such that \( P \vdash (\text{sxq } Q\theta \rightarrow t) \) for some pattern \( t \). Then \( [Q]_k(F, [\theta(V)]) :=^* (F', L) \), for some forests \( F, F' \) and with \( L \) verifying \( t \in L \).

**Proof.** Observe that from \( P \vdash (\text{sxq } Q\theta \rightarrow t) \) and by Lemma 3 we have \( P \vdash (\text{sxq } Q\theta) \).

Then we prove by complete induction on the structure of \( Q \) that \( P \vdash (\text{sxq } Q\theta \rightarrow t) \) implies \( [Q]_k(F, [\theta(V)]) :=^* (F', L) \), for some forests \( F, F' \) and with \( L \) verifying \( t \in L \).

- \( Q \equiv () \). If the query is of this form the result is not applicable because there is not \( t \) verifying \( P \vdash (\text{sxq } Q\theta) \) (in fact () is not representable as term of type \( \text{sxq} \).
- \( Q \equiv Q_1 Q_2 \).

Any proof of \( P \vdash \text{sxq} (\text{comp } Q_1 Q_2)\theta \rightarrow t \) must stat by a (FA) CRWL reduction step (see Figure 3), which must use an instance either of the \( \text{sxq} \) (\( \text{comp } Q_1 Q_2 \) = \( \text{sxq} Q_1 \) or of the rule \( \text{sxq} (\text{comp } Q_1 Q_2) = \text{sxq} Q_2 \). Assume the first rule is used (analogous for the second one). Then the (FA) inference step has a premise proving \( P \vdash \text{sxq} Q_1 \theta \rightarrow t \).

We check that the induction hypothesis can be applied to \( Q_1 \) verifying that it satisfies the premises of the lemma.

- \( Q_1 \equiv Q'_1 \) is an XQ query.
- The set \( V \) is the same as in the case of \( Q \). This is a consequence of Lemma 1, considering that \( Q \equiv Q'_p \) and \( Q_1 \equiv Q'_{p-1} \). Then \( V = \text{Rel}(Q'_p) = \text{Rel}(Q'_{p-1}) \), because:

\[
\begin{align*}
\text{Rel}(Q'_p) & = \text{by Lemma 1} \\
\text{Rel}(Q'_p) \cup \text{Rel}((Q'_p)_{1_1}) & = \text{by Definition 3 rule 2 with } Q \equiv Q_1 \ Q_2 \\
\text{Rel}(Q'_{p-p}) \cup \text{Rel}(Q_1) & = \text{by Definition 2 rule 1} \\
\text{Rel}(Q'_{p-p}) \cup \emptyset & = \text{by Definition 3 rule 3} \\
\text{Rel}(Q'_{p-p}) & = \text{by Lemma 3}
\end{align*}
\]
Then, by the induction hypothesis
\[ [Q_1]_k(\mathcal{F}, [\theta(V)]) :=^* (\mathcal{F}_1, L_1), t \in L_1 \] (1)

In \text{XQ} : By \text{XQ}_4

\[ [Q]_k(\mathcal{F}, [\theta(V)]) := [Q_1]_k(\mathcal{F}, [\theta(V)]) \cup [Q_2]_k(\mathcal{F}, [\theta(V)]) \] (2)

Combining (2) and (1) and considering that := is normalizing, that is, 
\[ [Q_2]_k(\mathcal{F}, [\theta(V)]) :=^* (\mathcal{F}_2, L_2) \] for some normal form \((\mathcal{F}_2, L_2)\):

\[ [Q]_k(\mathcal{F}, [\theta(V)]) :=^* (\mathcal{F}_1 \cup \mathcal{F}_2, L_1 ++ L_2) \]

with \( t \in L_1 \).

- \( Q \equiv \text{for } X_i \text{ return } Q_2 \). Then any proof of \( \mathcal{P} \vdash \text{sxq } Q \theta \rightarrow t \) must start with a (FA) inference using the program rule

\[ \text{sxq } (\text{xfor } X_i \text{ } Q_1 \text{ } Q_2) = \text{sxq } Q_2 \leftrightarrow X_i = \text{sxq } Q_1 \]

Next we check that \( Q_1, Q_2 \) verify the lemma premises, and that hence it is possible to apply the induction hypothesis to both subqueries.

In the case of \( Q_1 \):
- \( Q_1 \equiv Q'_1 \).
- \( \text{Rel}(Q'_1) = \text{Rel}(Q'_1) = V \), as consequence of Lemma 1 and of Definition 2, in particular of rule 5.
- In the premises of the (FA) inference at the root can be found the proof \( \mathcal{P} \vdash \text{sxq } Q_1 \theta = \text{sxq } Q_i(X) \), which by Lemma 3 implies \( \mathcal{P} \vdash \text{sxq } Q_1 \theta \rightarrow \theta(X) \).

Then applying induction: \([Q_1]_k(\mathcal{F}, [\theta(V)]) :=^* (\mathcal{F}_1, L_1)\), with \( \theta(X_i) \in L_1 \).

In the case of \( Q_2 \):
- \( Q_2 \equiv Q'_2 \).
- \( \text{Rel}(Q'_2) = \text{Rel}(Q'_2) \cup \{ X_i \} = V \cup \{ X_i \} \), as consequence of Lemma 1 and of Definition 2, rule 6.
- In the premises of the (FA) inference at the root can be found the proof \( \mathcal{P} \vdash \text{sxq } Q_2 \theta \rightarrow t \).

Applying induction: \([Q_2]_{k+1}(\mathcal{F}, [\theta(V \cup \{ X_i \})]) :=^* (\mathcal{F}_2, L_2), t \in L_2 \).

In \text{XQ} , by \text{XQ}_5

\[ [Q] := \bigcup_{1 \leq j \leq |L_1|} [Q_2]_{k+1}(\mathcal{F}_1, [\theta(V) \cdot l_j]) = (\mathcal{F}', L') \]

where \( \theta(V) \cdot l_j \) indicates the concatenation of \( l_j \) at the end of the list \( \theta(V) \).

Then since \( \theta(X_i) \in L_1 \) we have that \( \theta(X_i) \) is one of these \( l_j \). Then the result follows observing that \( \theta(V \cup \{ X_i \}) = (\theta(X_1), \ldots, \theta(X_i)) = \theta(V) \cdot \theta(X_i) \), since then from \([Q_2]_{k+1}(\mathcal{F}, [\theta(V \cup \{ X_i \})]) :=^* (\mathcal{F}_2, L_2), t \in L_2 \), implies \( i \in L' \).

- \( Q \equiv \text{if } C \text{ then } Q_1 \). Similar to the previous case.
\(-Q \equiv X_i.\)

In TOY, the representation of this query in TOY will be \(Q_T \equiv xp (\text{var } X_i).\) Any proof for \(P \vdash sxq Q_T \rightarrow t\) must start with a (FA) inference using the program rule \(sxq (xp E) = sxqPath E.\) Therefore this inference has a premise proving \(P \vdash sxqPath (\text{var } X_i) \rightarrow t.\) This proof must use again the (FA) inference, this time applying the rule \(sxqPath (\text{var } X) = X.\) Therefore in the proof of this statement we find a proof for \(P \vdash \theta(X_i) \rightarrow t\), which by Lemma 3 implies that \(\theta(X_i) \equiv t.\)

In XQ, applying XQ6, \(\llbracket x_i \rrbracket_k (F, \theta(V)) := (F, [\theta(X_i)])\) and then \(t \in \theta(X_i)\) as expected.

\(-Q \equiv \lfloor x_i/axis :: \nu \rfloor.\) 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 TOY of \(\lfloor x_i/child :: \text{name} \rfloor \equiv \lfloor \text{nameT "name"} \rfloor.\) From the premise \(P \vdash (sxq Q \theta \equiv t)\) and by Lemma 3 there is a CRWL proof for \(P \vdash sxq Q \theta \rightarrow t.\) The proof must start applying a (FA) inference rule of CRWL, applying the first rule of \(sxq\) (that is, \(sxq (xp E) = sxqPath E\) see Figure 4), with a rule instance defined by the substitution \(\sigma = \{E \mapsto X_i :/ \text{child} .::: (\text{nameT "name"})\}.\) The step must be of the form:

\[
\begin{align*}
X_i \theta :/ \text{child} .::: (\text{nameT name}) & \rightarrow X_i \theta :/ \text{child} .::: (\text{nameT name}) \\
\text{sxqPath} (X_i \theta :/ \text{child} .::: (\text{nameT name})) & \rightarrow \text{sxq Q \theta} \rightarrow t
\end{align*}
\]

The proof of the first premise is a direct consequence of Lemma 3. 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 4), and with an instance given by the substitution \(\sigma = \{X \mapsto X_i, S \mapsto \text{nameT name}\}.\)

\[
\begin{align*}
X_i \theta & \rightarrow X_i \theta \\
\text{child} .::: (\text{nameT name}) & \rightarrow \text{child} .::: (\text{nameT name}) \\
(\text{child} .::: (\text{nameT name})) X_i \theta & \rightarrow t \\
\text{sxqPath} (X_i \theta :/ \text{child} .::: (\text{nameT name})) & \rightarrow \text{sxq Q \theta} \rightarrow t
\end{align*}
\]

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 .::: (rule \(F .::: G) X = G (F X)\), see Section 3) with instance \(\sigma = \{F \mapsto \text{child}, G \mapsto \text{nameT name}, X \mapsto X_i \theta\}.\)

The inference must be of the form:

\[
\begin{align*}
\text{child} & \rightarrow \text{child} \\
\text{nameT name} & \rightarrow \text{nameT name} \\
X_i \theta & \rightarrow X_i \theta \\
(\text{nameT name}) (\text{child} X_i \theta) & \rightarrow t \\
(\text{child} .::: (\text{nameT name})) X_i \theta & \rightarrow t
\end{align*}
\]
Now the proof of \((\text{nameT name}) \ (\text{child } X, \theta) \rightarrow t)\) corresponds to an application of function \(\text{nameT}\) with two arguments: \(\text{name}\) and \((\text{child } X, \theta)\).

The program rule for \(\text{nameT}\) is \(\text{nameT } S \ (\text{xmlTag } S \ \text{Attr } L) = \text{xmlTag } S \ \text{Attr } L\). In order to apply this rule the second argument of \(\text{nameT}\), \((\text{child } X, \theta)\), must be reduced to a pattern of the form \(\text{xmlTag \ name \ Attr \ L}\). We choose the substitution \(\sigma = \{S \mapsto \text{name}\}\). Then the FA step is of the form:

\[
\begin{align*}
\text{name} & \rightarrow \text{name} \\
\text{child } X, \theta & \rightarrow \text{xmlTag \ name \ Attr \ L} \\
\text{xmlTag \ name \ Attr \ L} & \rightarrow t
\end{align*}
\]

\((\text{nameT name}) \ (\text{child } X, \theta) \rightarrow t)\)

Since \(t\) is a total pattern, from Lemma \(3\) applied to the third premise we have \(t \equiv \text{xmlTag \ name \ Attr \ L}\), that is, \(t\) is the representation in \(\text{TOY}\) of an XML element with label \(\text{name}\). The second premise implies a proof for \(\text{child } X, \theta \rightarrow \text{xmlTag \ name \ Attr \ L}\) in \(\text{CRWL}\). Again the rule \(\text{FA}\) is applied, this time using the program rule \(\text{child} \ (\text{xmlTag } \text{Name'} \ \text{Attr'} \ L') = \text{member} \ L'\) (the variables have been renamed). This time the instance is \(\sigma = \{\text{Name'} \mapsto \bot, \text{Attr'} \mapsto \bot\}.\) The use of \(\bot\) indicates that this values are not relevant for the result; only the list of children \(L'\) is necessary. The FA step must be of the form:

\[
\begin{align*}
\text{X, \theta} & \rightarrow \text{xmlTag} \ \bot \ \bot \ L' \\
\text{member} \ L' & \rightarrow \text{xmlTag \ name \ Attr \ L} \\
\text{child } X, \theta & \rightarrow \text{xmlTag \ name \ Attr \ L}
\end{align*}
\]

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

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

In \(\text{XQ}\): Observe that \(X_i\) is a relevant variable in \(Q\) (all the free variables of a subquery of \(Q'\) must be relevant if \(Q'\) is correct). Applying \(\text{XQ}_5\) we have that

\[
[SX_i/\text{child} :: \text{name}]_k(F, \theta(V)) = (F, L'')
\]

Then we prove that \(t \in L''\). This holds because by \(\text{XQ}_5\), \(L''\) is the list of nodes \(v\) such that

i) \(\text{child}^F(t_i, v)\). By the lemma hypothesis \(t_i \equiv X_i, \theta\), and the children of \(X_i, \theta = \text{xmlTag } A \ B \ L'\) are the elements of \(L'\). Then from item 2. \(t \in L'\), and therefore it satisfies this condition.

ii) \(\text{Label name of } (v) = \text{name}\). Also by item 2. about the label of \(t\) is \(\text{name}\). Therefore \(t \in L''\) as indicated in the lemma.

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

\(\text{XQ}_5\) The condition about the order in the nodes in \(\text{XQ}_5\) is not included because it has no effect in the result.
Let \( P \) be the \( \mathcal{T}_O\mathcal{Y} \) program of Figure 4, \( Q \) an SXQ query, \( t \) a \( \mathcal{T}_O\mathcal{Y} \) pattern, and \( \theta \) a substitution such that \( P \vdash (sxq \ Q \theta \ = \ t) \) for some \( \theta \). Then \( \mathcal{Q}_\theta(\emptyset, \cdot) \vdash (F, L) \), for some forest \( F \), and \( L \) verifying \( t \in L \).

Proof. In Lemma 4 consider the position \( p \equiv \varepsilon \). Then \( Q' \equiv Q \), \( V = \emptyset \) and \( k = 0 \). Without loss of generality we can restrict in the conclusion to \( F = \emptyset \), because \( \theta(V) = \emptyset \) and therefore \( F \) is not used during the rewriting process. 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 \( \mathcal{T}_O\mathcal{Y} \) program can produce every answer obtained by the XQ operational semantics.

Lemma 5. Let \( P \) be the \( \mathcal{T}_O\mathcal{Y} \) program of Figure 4, \( Q' \) be a SXQ query and \( Q, p \) such that \( Q \equiv Q'|_p \). Define \( V = \text{Rel}(Q', p) \) (see Definition 3) and \( k = |V| \).

Suppose that \( \mathcal{Q}_\theta(\emptyset) \vdash (F, \pi) \) for some \( F \), \( \pi \). Then, for every \( a_j \), \( 1 \leq j \leq n \), there is a substitution \( \theta \) such that \( \theta(X_i) = e_i \) for \( X_i \in V \) and a CRWL-proof proving \( P \vdash sxq \ Q \theta \ = \ a_j \).

Proof. Due to the Lemma 3 it is enough to prove that \( P \vdash sxq \ Q \theta \rightarrow a_j \). By complete induction on the structure of \( Q \).

- \( Q \equiv () \). Applying the rule \textbf{XQ1}, \([ ()]_k(F, \pi_k) : = (F, [ \cdot ]) \). In this case, \( n = 0 \), and therefore the result trivially holds.

- \( Q \equiv \{ Q_1 \} \). In this case, by \textbf{XQ2}, \([ Q_1 Q_2 ]_k(F, \pi_k) : = [ Q_1 ]_k(F, \pi_k) \cup [ Q_2 ]_k(F, \pi_k) : = (F_1, \pi_{n_1}) \cup (F_2, \pi_{n_2}) \). Then \( a_j \) is either in \( \pi_{n_1} \) or in \( \pi_{n_2} \).

  - If \( a_j \) in \( \pi_{n_1} \). Then we consider the reduction \([ Q_1 ]_k(F, \pi_k) : = (F_1, \pi_{n_1}) \).

The set of variables of \( Q' \) that are relevant for \( Q_1 \) is denoted by \( \text{Rel}(Q', p \cdot 1) \), and \( \text{Rel}(Q', (p \cdot 1)) = V \), because:

\[
\begin{align*}
\text{Rel}(Q', (p \cdot 1)) & \quad = (\text{by Lemma 1}) \\
\text{Rel}(Q', p) \cup \text{Rel}(Q', 1) & \quad = (\text{because } Q_1 Q_2 \equiv Q \equiv [Q'_p]) \\
\text{Rel}(Q', p) \cup \text{Rel}(Q, 1) & \quad = (\text{by Definition 2, rule 3}) \\
\text{Rel}(Q', p) \cup \text{Rel}(Q, \varepsilon) & \quad = (\text{by Definition 2, rule 4}) \\
\text{Rel}(Q', p) \cup \emptyset & \quad = \\
\text{Rel}(Q', p) & \quad = \\
V & 
\end{align*}
\]

For all \( X_i \) in \( V \), \( \theta(X_i) = e_i \), and by induction hypothesis, for every \( a_j \), \( 1 \leq j \leq n_1 \), there is a CRWL-proof proving \( P \vdash sxq \ Q_1 \theta \rightarrow a_j \). Now, applying a variant of the the third rule of sxq (for instance, sxq \( (\text{comp } Q_1 \ Q_2') \) = sxq \( Q_1' \), see Figure 4), there is a CRWL proof for \( P \vdash sxq \ (\text{comp } (Q_1 Q_2)) \theta \rightarrow a_j \), using the substitution \( \sigma = \{ Q_1' \mapsto Q_1, Q_2' \mapsto Q_2 \} \cdot \theta \) to obtain the rule instance.

\[
\begin{align*}
\text{sxq } (\text{comp } (Q_1 Q_2)) \theta & \rightarrow (\text{comp } (Q_1' Q_2')) \sigma \\
\text{sxq } Q_1' \sigma & \rightarrow a_j
\end{align*}
\]
The CRWL-proof of the first premise is obtained from Lemma 3 since both sides are the same term due to the definition of $\sigma$. The second premise is the result we have obtained by induction hypothesis since $sxq Q_1' \sigma = sxq Q_1 \theta$.

- If $a_j$ in $n_{n_2}$. Analogously to the previous case, the induction hypothesis can be applied to $Q_2$, concluding that for every $a_j$, $1 \leq j \leq n_2$, there is some CRWL-proof proving $P \vdash sxq Q_2 \theta \rightarrow a_j$, and hence for $P \vdash sxq (\text{comp} (Q_1, Q_2)) \theta \rightarrow a_j$ using the fourth rule of $sxq$ (Figure 4). In both cases for all $a_j$, $1 \leq j \leq n$, there is a CRWL-proof proving

$$P \vdash sxq (\text{comp} (Q_1, Q_2)) \theta \rightarrow a_j$$

which proves the result.

- $Q \equiv \text{for } X_{k+1} \text{in } Q_1 \text{return } Q_2$.

In this case, by $XQ_3$,

$$\begin{align*}
\{sxq_{k+1} \text{ in } Q_1 \text{ return } Q_2\}_k (F, r_k) &:= \bigcup_{1 \leq i \leq m} [Q_2]_{k+1} (F', r_k, \cdot, l_i) := * (F_1 \cup \ldots \cup F_m, \pi_{n_1}^1 + \ldots + \pi_{n_m}^m) \\
\{Q_1\}_k (F, r_k) &:= (F', l_m).
\end{align*}$$

Consider any $t \in \pi_{n_1}^1 + \ldots + \pi_{n_m}^m$, then $t \equiv a_s^r$, with $r \in \{1, \ldots, m\}$, $s \in \{1, \ldots, m_r\}$.

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 this set is identical to $V$, because:

$$\begin{align*}
Rel(Q', (p \cdot 1)) &= (\text{by Lemma 1}) \\
Rel(Q', p) \cup Rel(Q'_{p'}, 1) &= (\text{because } Q \equiv \{Q'_{p'}\}) \\
Rel(Q', p) \cup Rel(Q, 1) &= (\text{by Definition 2 rule 3}) \\
Rel(Q', p) \cup Rel(Q, \epsilon) &= (\text{by Definition 2 rule 3}) \\
Rel(Q', p) \cup \emptyset &= (\text{by Definition 2 rule 8}) \\
V &= (\text{by Definition 2 rule 8})
\end{align*}$$

By induction hypothesis there is a CRWL-proof proving $P \vdash sxq Q_1 \theta_1 \rightarrow l_r$ for some substitution $\theta_1$ such that $\theta_1(X_i) = e_i$, for $i = 1 \ldots m$, with $X_i \in V$. In particular $\theta_1(X_r) = e_r$.

- Consider the reduction $\{Q_2\}_{k+1} (F', r_k \cdot l_r) := * (F_r, \pi_{m_r}^r)$. We have seen already that $t \in \pi_{m_r}^r$. Let $V' = Rel(Q', (p \cdot 2))$ be the set of variables of $Q'$ that are relevant for $Q_2$. Then $V' = V \cup \{X_{k+1}\}$, because:

$$\begin{align*}
Rel(Q', (p \cdot 2)) &= (\text{by Lemma 1}) \\
Rel(Q', p) \cup Rel(Q'_{p'}, 2) &= (\text{because } Q \equiv \{Q'_{p'}\}) \\
Rel(Q', p) \cup Rel(Q, 2) &= (\text{by Definition 2 rule 3}) \\
Rel(Q', p) \cup \{X_{k+1}\} \cup Rel(Q, \epsilon) &= (\text{by Definition 2 rule 3}) \\
Rel(Q', p) \cup \{X_{k+1}\} \cup \emptyset &= (\text{by Definition 2 rule 8}) \\
Rel(Q', p) \cup \{X_{k+1}\} &= (\text{by Definition 2 rule 8}) \\
V \cup \{X_{k+1}\}
\end{align*}$$
By induction hypothesis, there is a CRWL-proof proving $P \vdash sxq Q_2\theta_2 \to t$ for some $\theta_2$ such that $\theta_2(X_j) = \theta_1(X_j) = e_j$, $1 \leq j \leq k$ and $\theta_2(X_{k+1}) = l_r$. Then we can define $\theta = \theta_1 \cup \theta_2$ without ambiguity, because $\text{dom}(\theta_1) \cap \text{dom}(\theta_2) = V$ and $\theta_2(X) = \theta_1(X)$ for every $X \in V$. Observe that in Section 2 we assumed that every for introduces a new variable, although variables in different scopes can share indexes like in the query for $X_1$ in ((for $Y_2$ in ...) (for $Z_2$ in ...)) ....

Now we can use a variant of the the fifth rule of sxq (see Figure 4) such as $sxq (\text{xfor } X Q_1' Q_2') = sxq Q_2' \iff X = sxq Q_1'$, and a substitution $\sigma = \{x \mapsto X_{k+1}, Q_1' \mapsto Q_1, Q_2' \mapsto Q_2\} \cdot \theta$, and build a CRWL-proof for $P \vdash (sxq (\text{xfor } X_{k+1} Q_1 Q_2))\theta \to t$ starting with a (FA) inference of the form:

$$
\frac{(xfor X_{k+1} Q_1 Q_2)\theta \to (xfor X Q_1' Q_2')\sigma}{sxq Q_1'\sigma = X\sigma} \frac{(sxq Q_2')\sigma \to t}{(sxq (xfor X_{k+1} Q_1 Q_2))\theta \to t}
$$

which can be rewritten as

(1) $(xfor X_{k+1} Q_1 Q_2)\theta_2 \to (xfor X_{k+1} Q_1 Q_2)\theta_2$
(2) $sxq Q_1\theta_1 = X_{k+1}\theta_2$
(3) $sxq Q_2\theta_2 \to t$

$t = (sxq (xfor X_{k+1} Q_1 Q_2))\theta_2 \to t$

taking into account the definition of $\sigma$ and $\theta$.

Now we check that the three premises can be proven in CRWL.

1. $P \vdash (xfor X_{k+1} Q_1 Q_2)\theta_2 \to (xfor X_{k+1} Q_1 Q_2)\theta_2$. Holds by Lemma 3.

2. $P \vdash sxq Q_1\theta_1 \iff X_{k+1}\theta_2$. Considering that $X_{k+1}\theta_2 = l_r$, and by Lemma 3 we must find a proof for $P \vdash sxq Q_1\theta_1 \to l_r$, and such proof exists by induction hypothesis.

3. $P \vdash sxq Q_2\theta_2 \to t$. Holds by induction hypothesis.

Observe that in fact $\theta_2$ contains also $X_{k+1}$ in its domain, $X_{k+1} \notin V$, but it still verifies the requirements of the Lemma, because $\theta_2(X_i) = e_i$ for $X_i \in V$.

- $Q \equiv \text{if } Q_1 \text{then } Q_2$. In $\text{TOY} : \text{xif (cond } Q_1\text{) } Q_2$.

In this case, by $XQ6$,

$$[[\text{if } Q_1 \text{ then } Q_2]]_k(F, \bar{v}) := \text{if } \pi_2([[Q_1]]_k(F, \bar{v})) \neq \bot \text{ then } [[Q_2]]_k(F, \bar{v}) \text{ else } (F, \bot)$$

We distinguish two cases:

- $[[\text{if } Q_1 \text{ then } Q_2]]_k(F, \bar{v}) := (F, \bot)$
  In this case, $[[\text{if } Q_1 \text{ then } Q_2]]_k(F, \bar{v}) = (F, \bot)$. Therefore, $n = 0$ and the result trivially holds.

- $[[\text{if } Q_1 \text{ then } Q_2]]_k(F, \bar{v}) := (F', \bar{v}')$
  In this case, the condition $Q_1$ returns some result, that is, $[[Q_1]]_k(F, \bar{v}) :=* \ldots$
\((\mathcal{F}'', \bar{b}_m)\) with \(m \neq 0\). Let \(a_j\) be any value in \(\bar{\pi}_n\). Then we prove that 
\(\mathcal{P} \vdash (\text{sxq } Q)\theta \rightarrow a_j\) for some \(\theta\) with \(\theta(X_p) = e_p\) for every \(1 \leq p \leq k\). 
The set of variables of \(Q'\) that are relevant for both \(Q_1\) and \(Q_2\) is identical to \(V\) (we skip the proof, analogous to the previous cases).

Hence the induction hypothesis can be applied to both \(Q_1\) and \(Q_2\).

* For \(b_1\), a substitution \(\theta_1\) such that \(\theta_1(X_i) = e_i\) for \(X_i \in V\), and a CRWL proof proving \(\mathcal{P} \vdash (\text{sxq } Q_1)\theta_1 \rightarrow b_1\).

* For \(a_j\), there is a substitution \(\theta_2\) such that \(\theta_2(X_i) = e_i\) for \(X_i \in V\), and a CRWL proof proving \(\mathcal{P} \vdash (\text{sxq } Q_2)\theta_2 \rightarrow a_j\).

Observe that we are assuming that each query introduces new variable names (Section 2). Then we can define \(\theta = \theta_1 \cup \theta_2\) without ambiguity.

Now, applying a variant of the seventh rule of \(\text{sxq}\) (for instance \(\text{sxq } \langle \text{xif } (\text{cond } Q_1') \ Q_2' \rangle = \text{sxq } \langle Q_2' \ \langle \theta \ angle \ 	ext{sxq } Q_1' \ angle \equiv A\), see Figure 4), and defining a substitution \(\sigma = \{Q_1' \rightarrow Q_1, Q_2' \rightarrow Q_2, A \rightarrow b_1\} \cdot \theta\) (in fact \(A\) can be bound to any \(b_i\), \(1 \leq i \leq m\)), we can build a CRWL proof for \(\mathcal{P} \vdash (\text{sxq } Q)\theta \rightarrow a_j\):

\[
\begin{align*}
\text{(sxq xif cond Q}_1 \ Q}_2)\theta & \rightarrow (\text{xif cond Q}_1') \ Q_2')\sigma \\
\text{(sxq Q}_1')\sigma & = A\sigma \\
\text{(sxq Q}_2')\sigma & \rightarrow a_j
\end{align*}
\]

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

\[
\begin{align*}
\text{(sxq xif cond Q}_1 \ Q}_2)\theta & \rightarrow (\text{xif cond Q}_1 \ Q}_2)\theta \\
\text{sxq Q}_1\theta_1 & = b_1 \\
\text{sxq Q}_2\theta_2 & \rightarrow a_j
\end{align*}
\]

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 3. The same Lemma indicates that proving \((\text{sxq } Q_1')\theta_1 \equiv b_1\) is equivalent to proving \(\text{sxq } Q_1 \theta_1 \rightarrow b_1\), and we have seen that it holds by induction hypothesis. The same happens with the third premise.

\(- \ Q \equiv \text{if } (\$x_i := \$x_j) \text{then } Q_2.
\]

In \(\text{TOY}\): \text{xif } (\text{xp } (\text{var } X_i) := \text{xp } (\text{var } X_j)) \ Q_2.

In this case, by \(\text{XQ}_6\),

\[
[\text{if } (\$x_i := \$x_j) \text{ then } Q_2]_k(\mathcal{F}, \pi) := \\
[Q_2]_k(\mathcal{F}, \pi) \quad \text{if } \pi_2([\$x_i := \$x_j]_k(\mathcal{F}, \pi)) \neq [\ ] \\
(\mathcal{F}, [\ ]) \quad \text{e.o.c}
\]

By \(\text{XQ}_7\), \(\[\$x_i := \$x_j\]_k(\mathcal{F}, \pi_k) := (\mathcal{F}', [a])\), if \(e_i = e_j\) with a some XML element, or \((\mathcal{F}', [\ ]))\), if \(e_i \neq e_j\). The last case corresponds to \(n = 0\) and it is trivial. Thus we assume that \(e_i = e_j\), and that \(\[Q_2\]_k(\mathcal{F}, \pi) := (\mathcal{F}', \pi_n)\). Let \(a_j\) be any element of \(\pi_n\). We must prove that \(\mathcal{P} \vdash \text{sxq } Q\theta \rightarrow a_j\) for some substitution \(\theta\) such that \(\theta(X_i) = e_i\) for every \(X_i \in V\).
In first place it is possible to check that \( P \vdash (sxq (xp (\text{var } X_i)) =_{\theta} sxq (xp (\text{var } X_j))) \) with \( \theta \) such that \( \theta(X_i) = e_i, \theta(X_j) = e_j \),

\[
\frac{(X_i)\theta \rightarrow t}{(sxq\,\text{Path}\,(\text{var } X_i)\theta) \rightarrow t}
\]

\[
\frac{(X_j)\theta \rightarrow t}{(sxq\,\text{Path}\,(\text{var } X_j)\theta) \rightarrow t}
\]

\[
\frac{(sxq\,(xp\,(\text{var } X_i)\theta)) \rightarrow t}{(sxq\,(xp\,(\text{var } X_j)\theta)) \rightarrow t}
\]

with \( t \equiv e_i \equiv e_j \), using the inference rules (JN), (FA) using the first rule of \( sxq \), (FA) using the first rule of \( sxq\text{Path} \) and finally proving the premises on top applying the Lemma 3.

The set of variables of \( Q_2 \) is identical to \( V \), and the induction hypothesis can be applied to \( Q_2 \) as in the previous case. Now, applying the sixth rule of \( sxq \) \( (sxq \,(x\text{if } Q_{11} := Q_{12}) \, Q_2) = sxq \, Q_2 \iff sxq \, Q_{11} == sxq \, Q_{12} \), see Figure 4), it is possible to find a CRWL proof for \( P \vdash (sxq \, Q)\theta \rightarrow e_i \) (the details are similar to the previous cases).

\( Q = [X_i] \).

In this case, \( [X_i]_{\theta}(F, r_k) := (F, [e_i]) \) by \( XQ_6 \). Then by Lemma 2, \( X_i \in V \) because \( X_i \) is free in \( Q \). \( \theta \) must be a substitution such that \( (X_i)\theta = e_i \).

The representation in \( TOY \) of \( x_i \) is \( sxq\,\text{Path}\,(\text{var } X_i) \), see Figure 4), and applying the first rule of \( sxq\text{Path} \), \( (sxq\,\text{Path}\,(\text{var } X) = X \), see Figure 4), we can build a CRWL proof for \( P \vdash (sxq \, Q)\theta \rightarrow e_i \) with instance \( \sigma = \{ X \rightarrow e_i \} \).

\[
\frac{\text{(var } X_i\theta \rightarrow \text{(var } X\sigma) \quad (X_i)\sigma \rightarrow e_i}{\text{sxq\,\text{Path}\,(var } X_i)\theta \rightarrow e_i}
\]

Applying the definition of \( \theta \) and \( \sigma \) in the premises we have:

\[
\frac{\text{(var } X_i\theta \rightarrow \text{(var } X\sigma) \quad (X_i)\sigma \rightarrow e_i}{\text{sxq\,\text{Path}\,(var } X_i)\theta \rightarrow e_i}
\]

and all the premises are consequence of Lemma 3.

\( Q = [x_i/\text{axis :: } \nu] \). The proof in this case is very similar to the corresponding case in Lemma 4 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 \( TOY \) program of Figure 4. Let \( Q \) be a SXQ query and suppose that \( [Q]_{k}(\emptyset, [\emptyset]) := (F, \bar{a}_n) \) for some \( F, \bar{a}_n \). Then for every \( a_j \), \( 1 \leq j \leq n \), there is \( P \vdash (sxq \, Q)\theta \equiv a_j \) for some substitution \( \theta \).

**Proof.** As in Theorem 1 suppose \( p \equiv \varepsilon \) and thus \( Q' \equiv Q \). Then \( V = \emptyset \) and \( k = 0 \). Then, if \( [Q]_0(\emptyset, [\emptyset]) := (F, \bar{a}_n) \) it is easy to check that \( [Q]_0(F', \emptyset) := (F, \bar{a}_n) \) for any \( F' \). Then the conclusion of the lemma is the same as the conclusion of the Theorem.

The proofs of the Lemmata 4 and 5 can be found in [2].
5 Application: Test Case Generation

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

Example 5. 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:

\[
Q = \text{for } b \text{ in doc("bib.xml")/bib/book, for } r \text{ in doc("reviews.xml")/reviews/entry, where } b/\text{title} = r/\text{title, for } \text{booktitle in } r/\text{title, revtext in } r/\text{review, publisher in } r/\text{publisher return } <\text{rev}> \text{booktitle publisher revtext } </\text{rev}>
\]

However, there is an error in this query, because in the statement for \$publisher in \$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 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 parse_xquery \( Q \) of query \( Q \) into TOY.
2. Let \( F_1, \ldots, F_k \) be the names of the XML documents occurring in \( Q' \). That is, for each \( F_i, 1 \leq i \leq k \), there is an occurrence of an expression of the form load_xml_file(F_i) in \( Q' \). (which corresponds to expressions doc(F_i) in \( Q \)). Let \( G \) be the result of replacing each doc(F_i) expression by a new variable \( D_i \), for \( i = 1 \ldots k \).
3. Let “expected.xml” be a document containing an expected answer for the query \( Q \).
4. Let \( G' \) be the result of replacing \( R \) by (load_doc "expected.xml")
5. Try the goal \( G' \), write_xml_file \( D_1 \) \( F'_1 \), \ldots, write_xml_file \( D_k \) \( F'_k \)

The idea is that the goal \( G == (\text{load_doc "expected.xml"}) \) looks for values of the logical 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 5.
prepareTC (xp E) = (xp E',L)
    where (E',L) = prepareTCPPath 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
prepareTCPPath (var X) = (var X, [])
prepareTCPPath (X :/ S) = (X :/ S, [])
prepareTCPPath (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. 5. TOY transformation rules for SXQ

The code uses the list concatenation operator ++ which is defined in 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 5.

Example 6. Consider the query of Example 5, and suppose the user writes the following document “expected.xml”:

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

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

```
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:
By comparing the test-case “revtc.xml” with the file “reviews.xml” we observe that the publisher is not in the 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 paper shows the embedding of a fragment of the XQuery language for querying XML documents in the functional-logic language TOY. Although only a small subset of XQuery consisting only of for, where/if and return statements has been considered, the users of TOY can now perform simple queries typical of database queries such as join operations. The embedding has respected the declarative nature of 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.

An extended version of this paper, including the proofs of the theoretical results and more detailed explanations about how to install TOY and run the prototype can be found in [2].

The most obvious future work would be introducing the let statement, 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 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 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 [5] and its encapsulated-search [10], or even the non-declarative collect primitive included in 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 [14] that would be applied to this setting.
References


