Untyped XQuery Canonization

Nicolas Travers[1], Tuyêt Trâm Dang Ngoc[2], and Tianxiao Liu[3]

[1] PRiSM Laboratory-University of Versailles, France. Nicolas.Travers@prism.uvsq.fr
[2] ETIS Laboratory - University of Cergy-Pontoise, France.
Tuyet-Tram.Dang-Ngoc@u-cergy.fr
[3] ETIS Laboratory - University of Cergy-Pontoise, France. Tianxiao.Liu@u-cergy.fr

Abstract. XQuery is a powerful language defined by the W3C to query XML documents. Its query functionalities and its expressiveness satisfy the major needs of both the database community and the text and documents community. As an inconvenient, the grammar used to define XQuery is thus very complex and leads to several equivalent query expressions for one same query. This complexity often discourages XQuery-based software developers and designers and leads to incomplete XQuery handling.
Works have been done in [DPX04] and especially in [Che04] to reduce equivalent forms of XQuery expressions into identified "canonical forms". However, these works do not cover the whole XQuery specification.
We propose in this paper to extend these works in order to canonize the whole untyped XQuery specification.

Keywords: XQuery evaluation, canonization of XQuery, XQuery processing

1 Introduction

The XQuery [W3C05] query language defined by the W3C has proved to be an expressive and powerful query language to query XML data both on structure and content, and to make transformation on the data. In addition, its query functionalities come from both the database community, and the text community. From the database languages, XQuery has inherited from all data manipulation functionalities such as selection, join, ordering, set manipulation, aggregation, nesting, unnesting, ordering and navigation in tree structure. From the document community, functions as text search, document reconstruction, structure and data queries have been added.

The XQuery query language is expressed using the famous FLWOR (FOR *...exp...* LET *...exp...* WHERE *...exp...* ORDER *...exp...* RETURN*...exp...*) expression form. But this simple form is not so simple: thus, any expression *exp* can also be recursively a FLWOR expression but also a full XPath expression.

In Table 1, Query A is a complex XQuery expression that defines a function that selects `books` with constraints on `price`, `keywords` and `comments` and that returns `price` and `isbn` depending on the number of returned `titles`. This query contains XPath Constraint, Filter, Quantifier, Document construction, Nesting, Aggregate, Conditional and Set operation, Ordering, Sequence and Function.

However, by using XQuery specifications, some expressions are found to be equivalents (ie. give the same result independently of the set of input documents). Thus, the Query B in Table 1 is an equivalent form of the previous Query A.

Query A

```
declare function local:f($doc as xs:string) as element()
{
    for $x in
        (doc("rev.xml")/review|doc("$doc")/catalog) [.
        contains("Robin Hobb")]/book/[.//price > 15]
    where some $y in $x/comments
        satisfies contains ($y, "Excellent")
    order by $x/@isbn
    return
        <book>
            {$x/@isbn}
            <price>{$x//price/text()}</price>
            {
                if (count($x/title) > 2)
                then {for $z in doc("books.xml")/book
                    where $z/@isbn = $x/@isbn
                    return
                            <title>{($z/title)[3]}</title>}
                else <title/>
            }
        </book>
}
```

Query B

```
declare function local:f($doc as xs:string) as element()
{
    let $l1 := for $f1 in doc("rev.xml")/review
        for $f2 in doc("$doc")/catalog
        return ($f1 | $f2)
    for $f3 in $l1
    for $x in $f3/book
    let $l2 := for $y in $x/comments
        where contains ($y, "Excellent")
        return $y
    let $l3 := orderby ($x, $x/@isbn)
    for $ordered in $l3
    let $l4 := count ($ordered/title)
    let $l5 := for $z in doc("books.xml")/book
        let $l6 := $z/title
        where
            $z/@isbn = $ordered/@isbn
            and $z/position () == 3
        return <title>{$l6}</title>
    where
        contains($f3, "Robin Hobb")
        and $x//price > 15
        and count ($l2) > 0
    return
        <book>
            {$ordered/@isbn}
            <price>{$ordered//price/text()}</price>
            {
                if ($l4 > 2)
                then {$l5}
                else <title/>
            }
        </book>
}
```

Table 1. Two equivalent XQuery queries

XQuery can generate a large set of equivalent queries. In order to simplify XQuery queries studies, it is useful to identify sets of equivalent queries and associate them with a unique XQuery query called : *Canonical query*. This decomposition is used in our evaluation model called TGV [TDL06,TDL07] in which each canonized expression generates a unique pattern tree. This paper aims at allowing all XQuery representation by adding missing canonization rules (not studied in [Che04] and [OMFB02]).

The rest of this paper is organized as follows. The next section describes related works, especially canonical XQuery introduced by [Che04]. Section 3 focuses on our extension of [Che04]'s work to the canonization of the full untyped XQuery. Section 4 reports on validation of our canonization rules and finally, section 5 concludes.

2 Related Work

2.1 GALAX

GALAX [FSC+03] is a navigation-based XQuery processing system. It has first propose a full-XQuery support by rewriting XQuery expression in the XQuery

core using explicit operation. The major issue of the navigational approach is to evaluate a query as a series of nested loops, whereas a more efficient evaluation plan is frequently possible. Moreover, the nested loop form is not suitable in a system using distributed sources and for identifying dependencies between the sources.

2.2 XPath

[OMFB02] proposes some equivalence between XPath axes. Those equivalences define XPaths in a single form with child and descendant expressions. Each "*or-self*" axis is bound to a union operator. A "*Parent*" or "*Ancestor*" axis is bound to a new variable with an "*exist()*" function a child/descendant. Table 2 illustrates some canonization of XPath axis.

XPath with specific axis	**Canonized XPath**
for $i in //a/parent::b	for $i in **//b** where **exists ($i/a)**
for $i in //a/ancestor::b	for $i in **//b** where **exists ($i//a)**
for $i in //a/descendant-or-self::b	for $i in //a**(//b | /.)**
for $i in //a/ancestor-or-self::b	for $k1 in **//b** **for $k2 in $k1//a** for $i in **($k1 | $k2)**

Table 2. XPath canonization

2.3 NEXT

Transformation rules suggested by [DPX04] are based on queries minimization of [AYCLS01] and [Ram02] in NEXT. They take as a starting point the *group-by* used in the OQL language, named *OptXQuery*. In order to eliminate redundancies while scanning elements, NEXT restructures the requests more efficiently to process nested queries. We do not take into account those transformation rules since [Che04] proposes transformation rules that create "*let*" clauses (and not a *group by* from OQL).

2.4 GTP

Works on GTP [Che04] propose transformation rules for XQuery queries. Aiming at structuring queries, XQuery queries are transformed in a canonical form of XQuery. The grammar of canonical queries is presented in table 3. This form is more restricted than XQuery specifications, but it allows us to cover a consequent subset of XQuery.

expr ::= (for $$fv_1$ in $range_1$, ... , $$fv_m$ in $range_m$)?
(let $$lv_1$:= "(" $expr_1$ ")", ... , $$lv_n$:= "(" $expr_n$ ")")?
(where φ)?
return <result>
$< tag_1 >\{arg_1\}< /tag_1 > ... < tag_n >\{arg_n\}< /tag_n >$
< /result>

Table 3. Canonical XQuery in GTPs

Thus, we obtain a specific syntax that enables us identifying XQuery main properties. These canonized queries must match the following requirements:

- XPath expressions should not contain building filters.
- *expr* expressions are XPaths or canonical XQuery queries.
- Expression φ is a Boolean formula created from a set of atomic conditions with XPaths and constants values.
- Each *range* expression must match the definition of a field of value.
- Each *range* expression is an XPath or an aggregate function.
- Each aggregate function can be only associated to a `let` clause.

In [Che04], it is shown that XQuery queries can always be translated into a canonical form. Lemmas enumerated below show canonical transformation rules.

1. XPath expressions can contain restrictions included in filters (between "[ ]"). With XQuery specifications, those filters can be replaced by defining new variables that are associated with predicate(s) (within the filter) into the *where* clause. Table 4 illustrates a transformation of a filter.

XQuery query	Canonized form
for $i in doc("cat.xml")/catalog/book [@isbn="12351234"]/title return {$i}	for $j in doc("cat.xml")/catalog/book **for $i in $j/title** where **$i/@isbn = "12351234"** return {$i}

Table 4. Query with filters

2. A FLWR expression with nested queries can be rewritten into an equivalent expression in which FLWR expressions are declared in *let* clauses. The new declared variable is used instead of the nested query. An example given in table 5 redefined a nested query in the *let* clause: "*let $l: = (...)*", and the return value becomes *$t*.

XQuery query	Canonized form
for $i in doc("cat.xml")/catalog/book return <book> {for $j in $i/title return {$j}} </book>	for $i in doc("cat.xml")/catalog/book **let $l :=** (for $j in $i/title return {$j}) return <book>**{$l}**</book>

Table 5. Nested queries transformation

3. A FLWR expression with a quantifier "every" can be transformed into an equivalent one using an expression of quantity. XQuery syntax defines quantifiers `every` as a predicate associated to the Boolean formula φ. The quantifier checks if each selected tree verifies the predicate. Table 6 returns all books for which all prices which are strictly higher than 15 euros. In order to simplify and to canonize this query, the "let" clause is created, containing books whose prices are lower or equal than 15 euros. If the number of results is higher than 0, then the selected tree ($i) does not satisfy the quantifier "every" and is not returned.

XQuery query	Canonized form
for $i in doc("cat.xml")/catalog/book where every $s in $i/price satisfies $s > 15 return {$i}	for $i in doc("cat.xml")/catalog/book **let $l :=**(for $j in $i/price where **$j <= 15** return {$j}) where **count($l) = 0** return {$i}

Table 6. Transformation of a quantifier "every"

4. In the same way, a FLWR expression, containing a quantifier "some", can be transformed. It is the same transformation, but the tree is selected if there is at least a tree that checks the condition (in the "let" clause).
5. Aggregates functions defined in FLWR expressions can be rewritten in "*let*" clauses, associated to a new variable. This variable replaces the aggregate function at the previous location.

Table 7 shows transformation of a nested query, an aggregate and a filter.

XQuery query	Canonized form
for $y in doc("rev.xml")/review [. contains ("daulphin")]/book where $y/price > 15 return <result> {$y/@isbn} {$y/price} <nb_titles>{ for $z in collection ("books")/book where $z/@isbn = $y/@isbn return {count ($z/title)} }</nb_titles> </result>	for $x in doc("rev.xml")/review, $y in $x/book let $l_1 := (for $z in collection ("books")/book let $l_2 := count ($z/title) where $z/@isbn = $y/@isbn return {$l_2}) where $x contains ("dauphin") and $y/price > 15 return <result> {$x/@isbn} {$y/price} <nb_titles>{$l1}</nb_titles> </result>

Table 7. Canonization of a nested query, an aggregate Function and a filter

As we can see, rules minimization [DPX04] and canonization [OMFB02] [Che04] helps at transforming XQuery queries into a canonical form. The [Che04] approach is more likely to deal with our needs, but it does not handle: **Ordering operators, Set operators, Conditional operators, Sequences** and **Functions declaration**.

Thus, we propose some more canonization rules in order to handle those XQuery requirements, making it possible to cover a more consequent set of the XQuery queries. Those new canonization rules will allow us to integrate those expressions in our XQuery representation model: TGV [TDL07] (Tree Graph View).

3 Canonisation

As said in the previous section, transformation rules transform a query into a canonical form. Since, it covers a subset of XQuery; we propose to cover much more XQuery queries. Thus, we add new canonization rules that handle all untyped XQuery queries.

In [Che04], five categories of expression are missing: ordering operators, set operators, conditional operators, sequences and function declaration. We thus propose to add canonization rules for each of those expressions.

3.1 Ordering (Order by)

Ordering classifies XML trees according to one or more given XPaths. The order of the trees is given by nodes ordering on values, coming from XPaths. This operation takes a set of trees and produces a new ordered set.

Lemma 3.1 : Ordering

An *XQuery* query containing an `Order By` clause can be transformed into an equivalent query without this clause. It is declared in a `let` clause with an aggregate function `orderby()` whose parameters are ordering fields with XPaths, and the ascending/descending sorting information. The `orderby` function results a set of sorted trees. The new linked variable replaces original used variables into the `return` clause. To keep the XML trees flow, a `for` clause is added on the given variable.

To obtain a canonical query, the `order by` clause must be transformed into a `let` clause. In fact, ordering is applied after `for`, `let` and `where` clauses, and before the `return` clause. Thus, results of preceding operations can be processed by the aggregate function: `orderby()`. This function orders each XML trees with a given XPath. Then, this aggregate function is put into a `let` clause, as specified in the canonical form. The new variable replaces all variables contained into the `return` clause.

Proof: Take a query Q. If Q does not contain an *orderby* clause, it is then canonical (for the order criteria).

Let us suppose that Q has n *orderby* clauses: *order by* $\$var_1/path_1, \$var_n/path_n$. Using the transformations lemmas on XPaths, $path_x$ are in a canonical form. The query Q is said to be canonical if the *orderby* clause is replaced by a *let* clause with an aggregate function *orderby*, and each transformed corresponding variable.

It is then necessary to study 3 cases of *orderby* clause:

1. If a variable is declared: *order by* $\$var_1/path_1$ *return* $\$var_1/path_2$, then: *let* $\$t := orderby\ (\$var_1, \$var_1/path_1)$ *return* $\$t/path_2$;
2. If two variables (or more) are declared, but identical: *order by* $\$var_1/path_1, \$var_1/path_2$ *return* $\$var_1/path_3$, then: *let* $\$t := orderby\ (\$var_1, \$var_1/path_1, \$var_1/path_2)$ *return* $\$t/path_3$;
3. If two variables (or more) are declared, but different: *order by* $\$var_1/path_1, \$var_2/path_2$ *return* $\{\$var_1/path_3, \$var_2/path_4\}$, then: *let* $\$t_1 := orderby\ (\$var_1, \$var_1/path_1), \$t_2 := orderby\ (\$var_2, \$var_2/path_2)$ *return* $\{\$t_1/path_3, \$t_2/path_4\}$.

Then, the $(n+1)^{th}$ *orderby* expressions in query Q can be written with n *orderby* expression, since a query with no *orderby* expression is canonical, then recursively, Q can be written without *orderby* clause.

Here is a example of an *orderby* clause canonization:

XQuery query	Canonized form
for $i in /catalog/book order by $i/title return $i/title	for $i in /catalog/book let $j := orderby ($i, $i/title) for $k in $j return $k/title

Table 8. *Orderby* canonization example

In table 8, the *for* clause selects a set of *book* elements contained in *catalog*. Then, it is sorted by values of the *title* element, and linked to the $j variable. The *orderby* clause canonization gives a *let* clause: $j, whose ordering function *orderby*() takes the variable $i for the input set, and $i/title to sort. The result set is then defined into the *for* clause ($k), in order to build a flow of XML trees. This new variable is used in the *return* clause by modifying XPaths ($k/title instead of $i/title).

Then, we obtain a canonized query without *orderby* clauses. This let clause creates a step of evaluation that would be easily identified in the evaluation process.

3.2 Set operators

Set operators express unions, differences or intersections on sets of trees. It takes two or more sets of trees to produce a single set. A *union* operator gathers all sets of trees, a *difference* operator removes trees of the second set from the first one and an *intersection* operator keeps only trees that exist in the two sets.

Lemma 3.2 : Set Operator

An XQuery query containing a set operator can be transformed into an equivalent query where the expression is decomposed and contains a *let* clause with two canonized expressions. The *return* clause contains the set operator between the two expressions.

Proof: Let's take a query Q. If the query Q does not contain a set operator between two FLWR expressions, then it is known as canonical.

When a query Q contains $n+1$ set operators between two expressions (other than variables), using canonization lemmas, we can say that this expressions are canonical. Let's take ξ, the set operator defined as $\{union, intersect, except\}$ (union, intersection, difference), then the table 9 illustrates the four possibilities of transformation:

Set expression	Canonized expression	Comments
$(expr_1\ \xi\ expr_2)$	let $\$t_3$:= for $\$t_1$ in $expr_1$ for $\$t_2$ in $expr_2$ return ($\$t_1\ \xi\ \$t_2$)	each expression is defined by a new variable. Those are linked by the operator.
$(expr_1\ \xi\ expr_2)/P$	let $\$t_3$:= for $\$t_1$ in $expr_1$ for $\$t_3$ in $expr_2$ return ($\$t_1\ \xi\ \$t_2$) ... $\$t_3/P$	The expression is broken up. 1) the set operator 2) the expression is replaced by the variable.
$\$XP(P_1\ \xi\ P_2)$	for $\$t_x$ in XP let $\$t_3$:= for $\$t_1$ in $\$t_x/P_1$ for $\$t_2$ in $\$t_x/P_2$ return ($\$t_1\ \xi\ \$t_2$)	A new variable is created. Apply the set operator (rule 1) on the new variable
$\$XP(P_1\ \xi\ P_2)/P_3$	for $\$t_x$ in XP let $\$t_3$:= for $\$t_1$ in $\$t_x/P_1$ for $\$t_2$ in $\$t_2/P_2$ return ($\$t_1\ \xi\ \$t_2$) ... $\$t_3/P$	Use the second and third decomposition rule on set expressions between XP et P_3

Table 9. Transformation of different set expressions

Thus, a query Q that contains $n+1$ set operators between two expressions can be rewritten with n set operators. If there are no set operators, it is canonical. Then, recursively, any query Q can be canonized without set operators.

Here a canonization example of a set expression:

XQuery query	Canonized form
for $i in (/catalog | /review)/book return $i/title	let i_3$:= for $$i_1$ in /catalog for $$i_2$ in /review return (i_1$ | i_2$) for $i in i_3$/book return $i/title

Table 10. Canonization of a set expression

In table 10, the *for* clause contains a union "|" between two sets. The first set is */catalog* and the second one */review*. On each one, the *book* element is selected. The title is then projected for each book. The canonization of the union operator (shortened "|") gives a *let* clause ($\$i_3$) containing two expressions $\$i_1$ and $\$i_2$. Each one is defined by a *for* clause on expected paths. The *let* clause $\$i_3$ returns the union of the two variables. Then, the XML trees flow is rebuilt by the *for* clause i_3 on the *book* element. We then obtain a canonized query where set operators are decomposed to detail each step of the procedure.

3.3 Conditional operators

Conditional operators bring operational processing on XML documents. Indeed, results of conditional operators depend on a given predicate. Then, the first result is returned if the constraint is *true*, the second one else. In the possible results, we can find XPath expressions, nested queries, tags or strings. In the case of nested queries, it is then necessary to canonize them to create a single canonized form.

Lemma 3.3 : Conditional Operators

An XQuery query containing a conditional operator (if/then/else) and a nested query, this one can be transformed into an equivalent query where the nested query will be declared in a clause *let*.

This lemma can be demonstrated in the same way of unnested queries [Che04] (section 2.4). Thus, recursively, we are being able to show that any query containing a nested query in a conditional operator can be canonized.

Here is a canonization example of a query with a conditional operator:

XQuery query	Canonized form
for $i in /catalog/book return {if contains ($i/author, "Hobb") then (for $j in $i//title return $j) else ($i/author)}	for $i in /catalog/book let $l := for $j in $i//title return $j return {if contains ($i/author, "Hobb") then ($l) else ($i/author)}

Table 11. Canonization example of conditional operators

In table 11, a conditional operator is declared in the *return* clause with a constraint on the author's name that must contain the word *Hobb*. If the word is contained, the nested query $j returns the *title*(s) of *book* else the *author* is returned. We obtain a canonized query where nested queries in conditional operators are set in a *let* clause.

3.4 Sequences

Sequences are sets of elements on which operations are applied. Indeed, when a constraint is applied on a sequence using brackets (*XPath*), the constraint is applied on the set of the trees defined by XPath (and not on each one). This operation gathers sets of trees in order to produce a unique set one which we apply the given constraint.

Lemma 3.4 : Sequences

An XQuery query containing a sequence can be rewritten in an equivalent query without sequences. Each sequence is translated in a *let* clause on which operations are put.

Sequences' filters behave like on current XPaths. They applied on results of the sequence. So, the proof is similar to the filter's one in lemma (2.3.1) of [Che04]. Sequences are built by grouping information. Thus any sequence expression is declared in a *let* clause, generating a new variable that could be used in the remaining query.

XQuery query	Canonized form
for $i in (/catalog/book)[2] return $i/title	let i_1$:= for $x in /catalog/book return $x for $i in i_1$ where $i/position() == 2 return $i/title

Table 12. Example of sequences canonization

In table 12, a sequence is defined in the for clause. The catalog's book set is aggregated. Then the second book element is selected (and not the second element of each set). Then, its title is projected. The canonization step produces a *let* clause in which the for clause is declared on required elements. Then, the new variable is used in the for clause $\$i$ with a constraint on position. Finally, the title is returned.

3.5 Functions

Function definition is useful to define a query that could be re-used many times, or to define queries with parameters. In XQuery, functions take parameters in input and a single set in output. Inputs and output are typed.

Lemma 3.5 : Functions

An XQuery function containing an XQuery expression can be rewritten in an equivalent function containing a canonical expression.

In Table 13, a function is defined (*local: section*) with a parameter in input. This input is defined by the for clause: *for $f in doc("catalog.xml")/catalog*, which set of trees will be used in the called function: *local:section ($f)*. In the function, each *book* element returns its title, and the set of all the titles contained in the sections (*$/section/title*). As we can see, the function contains a nested query. The

unnesting canonization step transforms the query into a canonized form inside the function.

XQuery query	Canonized form

declare function local:section
($i as element()) as element ()* {
for $j in $i/book
return
<book>
{$j/title}
{for $s in $i/section/title
return <section>
{$s/text()}
</section>}
</book>
}
for $f in doc("catalog.xml")/catalog
return local:section($f)
	declare function local:section
	($i as element()) as element ()*
	{
	for $j in $i/book
	let $l := (for $s in $i/section/title
	return <section>
	{$s/text()}
	</section>)
	return
	<book> {$j/title} {$l} </book>
	}
	for $f in doc("catalog.xml")/catalog
	return local:section($f)

Table 13. Function transformation

3.6 Canonical XQuery

Thus, using the previous lemmas and those proposed by [Che04], we can cover a broad set of expressions over XQuery. We can now cover: ① XPath expressions with filters ② *for*, *let* and *return* clauses ③ Predicates in the *where* clause ④ Nested queries ⑤ Aggregate functions ⑥ Quantifiers ⑦ Ordering operators ⑧ Set operators ⑨ Conditional operators ⑩ Sequences ⑪ Definition of functions. The only part of XQuery we do not consider yet is typing. Adding typing to the canonized form needs some works using XQuery/XPath typing consideration [GKPS05] on validated XML document.

Table 14 summarizes the additional canonization rules we propose. Those rules allow us to cover all untyped XQuery queries.

	Expressions	Canonical Form
R1	$order\ by\ var/xp$	$\Rightarrow let\ \$l_1 := orderby(var, var/xp)$
R2	$(expr_1$ union $expr_2)$	$\Rightarrow let\ \$i_3 :=$ for $\$i_1$ in $expr_1$, $\$i_2$ in $expr_2$ return $(\$i_1$ union $\$i_2)$
	$(expr_1$ intersect $expr_2)$	$\Rightarrow let\ \$i_3 :=$ for $\$i_1$ in $expr_1$, $\$i_2$ in $expr_2$ return $(\$i_1$ intersect $\$i_2)$
	$(expr_1$ except $expr_2)$	$\Rightarrow let\ \$i_3 :=$ for $\$i_1$ in $expr_1$, $\$i_2$ in $expr_2$ return $(\$i_1$ except $\$i_2)$
R3	$if\ expr_1$	$let\ \$l_1 := expr_2, \$l_2 := expr_3$
	$then\ expr_2$	$\Rightarrow if\ expr_1\ then\ \$l_1\ else\ \$l_2$
	$else\ expr_3$	(if each $expr_2$ and $expr_3$ are nested queries)
R4	$(expr_1)/expr_2$	$\Rightarrow let\ \$l_1 := expr_1$ ... $\$l_1/expr_2$

Table 14. Proposed canonization rules

Using all these rules, we can now deduce that the canonized form of Query A of Table 1 is the Query B of Table 1.

Theorem 3.1 : Canonization

All untyped XQuery queries can be canonized.

With all previous lemmas, we can infer theorem 3.1 that defines a grammar for canonical XQuery queries (Table 15). We can see that canonical queries start with a FLWR expression *Expr* and zero or more functions. The canonical form of *Expr* is composed of nested queries, aggregate functions, XPaths and non-aggregate functions. Moreover, set operators are integrated in these expressions, while the conditional operations are integrated into *ReturnClause*. The *Declaration* has also

a canonical form that prevents any nested expressions. XPaths do not contained anymore filters, sequences, nor set operators, since those are canonized.

XQuery ::=	(*Function*)* FLWR;
FLWR ::=	("for" "$" STRING " in " *Declaration* (, "$" STRING " in " *Declaration*)* | "let" "$" STRING ":=" "(" *Expr* ")" (, "$" STRING ":=" "(" *Expr* ")")*)+ ("where" *Predicate* (("and" | "or") *Predicate*)*)? "return " *ReturnClause* ;
ReturnClause ::=	"{" *CanonicExpr* "}" | "{" "if" *Predicate* "then" "(" *Expr* ")" "else" "(" *Expr* ")" "}" | "<" STRING ">" (*ReturnClause*)* "</" STRING ">" ;
Expr ::=	*FLWR* | "(" *Path SetOperator Path* ")" | *CanonicExpr* | *aggregate_function* ;
CanonicExpr ::=	*Path* | *non_aggregate_function*;
Declaration ::=	"collection" "(' " STRING " ')" (*XPath*)? | *CanonicExpr*;
Path ::=	"$" STRING *XPath* (EndXPath)?;
Predicate ::=	*Val Comp Val* | *QName* "(" ((*Val* ",")* *Val*)? ")";
Comp ::=	">" | "<" | "=" | "<=" | ">=" | "! =" ;
Val ::=	' STRING ' | Number | *Path*;
XPath ::=	("/" *Element* | "//" *Element*)+;
SetOperator ::=	"|" | "-" | "/" ;
EndXPath ::=	"/" (*Attribut* | *Element* | "text()");
Element ::=	(*QName* | "." | "..");
Attribute ::=	"@" *QName*;
QName ::=	(STRING ":")? STRING;
Function ::=	"declare function" QName "(" "$" STRING "as" "element" ("," "$" STRING "as" "element")* ")" "as" "element" "{" FLWR "}";

Table 15. Untyped Canonical XQuery

4 Validation

The use cases listed in Table 16 were created by the XML Query Working Group to illustrate important applications for an XML query language. Each use case is focused on a specific application area. Each use case specifies a set of queries that might be applied to the input data, and the expected results for each query. They are designed to cover the most part of XQuery specification.

Use case	Description	Specified	Recognized
"XMP"	Experiences and Exemplars	12	12
"TREE"	Queries that preserve hierarchy	6	6
"SEQ"	Queries based on Sequence	5	5
"R"	Access to Relational Data	18	18
"SGML"	Standard Generalized Markup Language	10	10
"STRING"	String Search	5	5
"NS"	Queries Using Namespaces	8	8
"PARTS"	Recursive Parts Explosion	1	1
"STRONG"	queries that exploit strongly typed data	12	0

Table 16. Use cases, number of specified queries, number of supported queries

Our canonization rules cover 8 of 9 use cases [DPD+05] of the W3C: XMP, TREE, SEQ, R, SGML, STRING, NS, PARTS. The use-cases category not covered by our canonization rules is the STRONG category that queries type information.

We have already implemented the XQuery canonization in our XML-based mediation system: XLive[TD07]. With the canonization, the XQuery processor has been easier to design and implement, and any untyped XQuery can be evaluated with XLive.

5 Conclusion

In this paper, we have extended the works of [OMFB02] and [Che04] in order to recognize the full untyped XQuery specification.

We claim that thanks to our canonization rules, all works that aim to manipulate XQuery could handle the full untyped XQuery specification with only a minimal XQuery subset to recognize. Especially for our TGV model [TDL06,TDL07] which is a simple translation from canonized XQuery queries [TDN07].

Adding typing to the canonized form needs some works using typing consideration [GKPS05] on validated XML document. We are currently working on this issue.

This work has been implemented as a module in the XLive mediation system that evaluates any XQuery query on distributed heterogeneous sources.

Acknowledgement This work is supported by the ACI Semweb and the ANR PADAWAN projects. The research prototype XLive system is an Open Source software and can be downloaded on : $http://www.prism.uvsq.fr/users/ntravers/xlive$

References

[AYCLS01] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *SIGMOD Conf.*, 2001.

[Che04] Z. Chen. From Tree Patterns to Generalized Tree Patterns : On Efficient Evaluation of XQuery, 2004.

[DPD+05] D.Chamberlin, P.Fankhauser, D.Florescu, M.Marchiori, and J.Robie. XML Query Use Cases, september 2005.

[DPX04] A. Deutsch, Y. Papakonstantinou, and Y. Xu. The NEXT Framework for Logical XQuery Optimization. In *VLDB*, pages 168–179, 2004.

[FSC+03] M.F. Fernández, J. Siméon, B. Choi, A. Marian, and G. Sur. Implementing xquery 1.0: The galax experience. In *VLDB*, 2003.

[GKPS05] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of XPath query evaluation and XML typing. *ACM (JACM)*, 52, 2005.

[OMFB02] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath : Looking Forward. In *Proc. of the EDBT Workshop on XML Data Management (XMLDM)*, volume 2490 of *LNCS*, pages 109–127. Springer, 2002.

[Ram02] P. Ramanan. Efficient Algorithms for Minimizing Tree Pattern Queries. In *ACM SIGMOD*, pages 299–309, June 2002.

[TD07] N. Travers and T.-T. Dang Ngoc. XLive : Integrating Source With XQuery. *WebIST*, March 2007.

[TDL06] N. Travers, T.-T. Dang Ngoc, and Tianxao Liu. TGV: an Efficient Model for XQuery Evaluation within an Interoperable System. *Interoperability in Business Information Systems (IBIS)*, December 2006.

[TDL07] N. Travers, T.-T. Dang Ngoc, and Tianxao Liu. TGV : a Tree Graph View for Modelling Untyped XQuery. *Database Systems for Advanced Applications (DASFAA international conference)*, April 2007.

[TDN07] N. Travers and T.-T. Dang-Ngoc. An extensible rule transformation model for xquery optimization. In *The 9th International Conference on Enterprise Information Systems (ICEIS)*, Madeira, Portugal, 2007.

[W3C05] W3C. An XML Query Language (XQuery 1.0), 2005.