ISSN 1346-5597

National Institute of Informatics

NII Technical Report

An XQuery Cost Model in Relative Form

Soichiro Hidaka, Hiroyuki Kato, and Masatoshi Yoshikawa

NII-2005-016E
Nov. 2005

URL: http://www.nii.ac.jp/ Tokyo 101-8430, JAPAN +81-3-4212-2000 FAX: +81-3-3556-1916

An XQuery Cost Model in Relative Form

Soichiro Hidaka*, Hiroyuki Kato*, and Masatoshi Yoshikawa**

* National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
FAX: +81-3-3556-1916
{hidaka,kato}@nii.ac.jp
** Graduate School of Information Science, Nagoya University,
Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan
FAX: +81-52-789-4383
yosikawa@is.nagoya-u.ac.jp

Abstract. We propose an XQuery cost model that is able to estimate the performance gain of source-level transformation. The cost of major language constructs, including FLWOR, quantified, path, element construction, and predicate expressions are captured. The evaluation of optimization using existing real engines suffer from problems, such as lack of applicability to other engines, instability that comes from evolution, difficulties in reproduction as a result of difficulties in acquisition, as well as unsuitability for evaluation of the optimization based on a static analysis in the absence of run-time information. This research is a first attempt to provide a virtual engine to facilitate the evaluation of various optimization techniques without introducing real engines. The cost model consists of simple recursive functions based on functional language constructs. They are determined using formal semantics and other known efficient algorithms. The model enables analytic comparison of costs between expressions before and after transformation in an engine-independent manner. An engine-specific evaluation strategy can be incorporated if necessary. We have succeeded in proving that various transformations, including auxiliary ones used in optimization are not cost-increasing, as well as in detecting cost-increasing transformations that should be avoided.

1 Introduction

When we mention optimizations based on program transformations, we should prove cost reduction as well as semantic equivalence.

Various transformation rules are applied in an attempt to reduce costs in rewriting-based optimization of query languages. These rules include not only transformations that obviously reduce costs, but also auxiliary transformations whose cost benefits may be non-trivial. If those transformations turn out to degrade the cost, they may hinder the total effectiveness of rewriting. Therefore, each transformation applied in the course of the optimization should be guaranteed not to increase cost. Consequently, estimating costs before and after transformations are crucial in determining optimization guidelines.

XQ := *literal*

(XQ)	/* parenthesized expr. */
XQ,XQ	/* sequence */
for $\$QName$ **in** XQ (**where** XQ)? **return** XQ	/* for expr. */
let $\$QName := XQ$ (**where** XQ)? **return** XQ	/* let expr. */
some $\$QName$ **in** XQ **satisfies** XQ	/* existential */
every $\$QName$ **in** XQ **satisfies** XQ	/* universal */
$\langle QName\rangle\{XQ\}\langle/QName\rangle$	/* direct elem. constructor */
element $QName$ $\{XQ\}$	/* computed elem. const. */
$XQ/QName$ | $XQ//QName$	/* relative path expr. */
$XQ[XQ]$	/* predicate expr. */
$\$QName$	/* variable reference */
if (XQ) **then** XQ **else** XQ	/* conditional expr */
XQ **or** XQ | XQ **and** XQ	/* logical expr. */
$/QName$ | $//QName$	/* absolute path expr. */
$QName(XQ,XQ,\ldots,XQ)$	/* function call */
$op(XQ,XQ,\ldots,XQ)$	/* other uninterpreted functions and n-ary operators */

Fig. 1. Abstract Syntax Tree of the XQuery subset discussed in this paper

One might verify those cost reduction using real-world engines. However, it is almost impossible to test all the existing engines. Moreover, it is not applicable to optimizations based on static analysis, in which run-time data cannot be investigated. Thus, some yardstick that is independent of the run-time information, such as data quantities, are required.

We propose a cost model for XML query language XQuery [1]. The cost model has granularity to the extent that it is able to capture the processing of major language constructs shown in Fig. 1, including FLWOR, quantified, and path expressions, while maintaining independence from specific engines. Please note that operators that are not explicitly listed can be modeled as uninterpreted functions, whose cost consists of the sum of the costs of arguments and the cost of the body itself.

Our cost model is mostly derived from formal semantics. Two exceptions are element construction and path expression. The former cost was not apparent in formal semantics, while the latter has an evaluation algorithm proposed in the literature that is more efficient than naively applying normalization rules in formal semantics. The cost models of those expressions are derived separately along with the verification of using real engines.

In this paper, we show the relative cost change by transformations applied in XQuery source level optimizations. For query rewritings that are composed only of cost-non-increasing transformations, the transformations on the whole can be inductively guaranteed to maintain or decrease costs.

Research that deals with the complexity of XQuery already exists [2], where the class of complexity is determined at the abstract level, such as in Turing machines or logic circuits. In such an abstraction level, one can not determine,

given two semantically equivalent XQuery codes, which is expected to run faster. From the complexity theoretical point of view, both of them fall into the same class, thus making no difference.

The cost model is intended to be used for optimizations based on static analysis, in which relative cost before and after the rewriting is estimated. Concrete values are unnecessary. Therefore, the cost of subexpressions that do not participate in the transformation can be safely left uninterpreted. Moreover, components of the costs of those subexpressions, namely the selectivity or the size can be determined in cooperation with other research contributions that actually calculate concrete values [3, 4].

We have determined, for each XQuery language construct, simple recursive cost functions that are expected to be satisfied. For example, in **for** expressions without a **where** clause, evaluation cost with respect to a **return** clause is determined by multiplying the cost of the **return** clause itself and the length of expressions as a sequence in the **in** clause.

Many equalities in source level transformations have been presented in the literature [5–9]. However, we were unable to find equalities that came along with the cost model at the granularity we are dealing with.

XQuery formal semantics [10] includes transformation rules that convert XQuery to its subset called XQuery Core. However, our transformation is wider in scope.

Besides the XQuery research area, we found the approach by Fegaras et al., who deal with the normalization rules, along with the cost model in the context of OQL [11], closest to ours. Although we found a strong affinity with our XQuery cost model, some problems did not allow us to adopt this approach directly. A detailed discussion is provided in section 2.4.

Once the cost model is determined, one can find out the specific transformations that should be avoided, which seems to be cost-preserving at first glance. For example, the result of an **and** expression can be determined (if we do not assume parallel processing) if either an operand first evaluated, turned out to be false. The evaluation of the other operand is not necessary. However, there are engines that always evaluate both operands (our experience can be seen in section 4.2). This difference in evaluation strategies gives us warnings when a transformation rule includes an **and** expression its result. Then we can choose either, (1) not to do such rewriting or (2) replace the **and** expression with an equivalent **if** expression.

The cost model provides an abstract estimation of the effectiveness of rewriting transformations. In the optimization research field, performance evaluation is often conducted by experiments using real engines. However, as they are conducted on their own engines [8][1], applicability to other engines is not guaranteed for the results. Since many of the engines are still in the research stage, stable reference engines available to everyone do not yet exist. In addition, these engines

[1] [5–7, 12] cited earlier propose transformation rules, but do not include the performance evaluation.

make constant progress, which makes these results quickly obsolete. Installations are also hard tasks.

This paper describes the first attempt to build a cost model that can serve as a *Virtual Engine* to facilitate the evaluation of optimizations. Just like Light XQuery [13] provides compact semantics sufficient for XQuery researches as a whole, this cost model provides "semantics in the cost domain" for XQuery source level optimization research. Indeed, this work is motivated by the desire fore supporting the equalities used in [14], on the cost side.

The rest of this paper is organized as follows. Section 2 gives general remarks on the rationale of the cost definition. Section 3 defines the concrete cost model. Cost changes caused by various transformations, including monad laws, are shown in section 4. Most of them are proved to either maintain or reduce costs. Section 5 refers related works, and finally the prospect of extending the cost model with respect to query size is described in section 6, followed by a conclusion and future work in section 7.

The transformation rules that appear in this paper are not exhaustive, because the purpose of this work is to provide cost formulas in relative form for major language constructs depicted in Fig. 1. The authors believe that most of the transformation rules that appear in the literature [5–9] can be captured by those presented in this paper. Transformations that are not covered in this paper include ones that are not cost-effective in terms of our XQuery source-to-source level optimization and conversely, our transformations that did not appear in the literature consist of either ones that are cost ineffective with respect to the underlying specific engines, or ones that involve intractable language components for these engines.

Although full-fledged FLWOR expressions or quantified expressions are not explicitly handled, cost models of these expressions with successive **in** clauses can be reduced to our model via the normalization rules in formal semantics [10]. In addition, cost changes of some of the other transformations related to join operations that were originated from [15] were left out, because these transformations can be represented by a composition of other transformations described in this paper. We plan to deal with a more complete set of transformation rules in a separate paper. The semantic equivalences of most transformations described here, unless explicitly mentioned, are proved in appendix A, all of which are easily derived from formal semantics.

This paper assumes a certain knowledge about XQuery. Readers unfamiliar with this query language may refer to introductory books such as [16, 17].

2 Rationale of cost model definitions

In this paper, weights of arbitrary XQuery expressions are statically estimated in order to evaluate their cost before/after transformation and quantify the effect of the optimization[2].

[2] Other possible approaches may include cost estimation based on actual test-executions like in Weight Finder [18].

Ultimately, concrete cost can be determined based on physical algorithm (plan) [11]. Costs estimated at the abstraction level of XQuery expressions may not suite the evaluation strategies in each real-world engine.

On the other hand, maintaining moderate abstraction levels, where no particular specialization in a specific engine is assumed, is essential in pursuing optimizations applicable to wide range of engines.

Here we aim at building cost models that do not remarkably depart from the physical plans. Verification using several existing engines are quoted where necessary.

2.1 Justification of cost definition

Cost models can be justified by the following conditions:

1. Procedures that the cost model assumes are shown to yield the same results as with XQuery (formal) semantics.(Required)

   Example: In cases of **for** expressions, at first the **in** clause is determined. The **return** clause is evaluated for each item of the resultant input sequence. Note that lazy evaluation is not exploited here. Therefore, if the **return** clause does not refer to the bound variable introduced in the **for** clause, it is redundant to evaluate it as many times as the length of the input sequence. However, it can be semantically justified in the sense that it is simulating an engine that does return valid results.

2. Avoid redundant computation.
   This condition should be necessary because the above condition alone is not always realistic.

   Example 1: Predicate of the **some** expressions are evaluated for all items in **in** clauses regardless of the initial value of the predicates, without stopping the evaluation even if one of the evaluations is found to yield true.

   Example 2: For the processing of an **order by** clause, produce an instance in which all the permutations for the input tuple stream are enumerated, and all the permutations are scanned to return only the permutation that satisfies the specification in the **order by** clause.
   Both of these examples satisfy condition 1 yielding valid results. However, these implementations are unrealistic from the standpoint of computational complexity.

To claim the adherence to condition 2, it is necessary to show that the cost model can simulate the behavior of a real engine that is considered useful (or conversely, it has to be shown that the real engine behaves similar to the cost model from the point of view of computational complexity).

For the cost of language constructs that are not intuitively apparent, we try to avoid the departure of the cost model from the real world by conducting a verification using an existing engine, such as Galax [19] or eXist [20].

2.2 Our cost definition strategy

Cost functions are mostly based on formal semantics [10].

However, Gottlob et al. [21]s combined (data and query) complexity is used for the path expression. To summarize,

Estimation based on formal semantics: Formal semantics are utilized for expressions like **for**, whose physical plan is expected to match formal semantics.

Estimation is based on the inference rules and judgments in formal semantics.

For each inference rule, the cost of the expression in the consequence part can be obtained by the sum of the cost of the expressions in the premise part.

Estimation based on preceding particular research: For the expressions whose costs can be calculated in prior research, the contribution is utilized.

- Gottlob et al. [21] have proposed an evaluation strategy in polynomial order for Core XPath.
  In this paper, behaviors of real engines are investigated for path expressions. For engines that can be considered to actually obey, Gottlob's polynomial model is adopted. Details are described in section 2.3.
- For monoid comprehension, which corresponds to FLWOR (excluding ordering portions) expression, Fegaras et al. [11] had proposed a simple cost model.
  Although some definitions are not intuitive, the method to recursively define the cost function using the size and selectivity function, have been adopted because we find an affinity with the functional aspect of the XQuery in the method. Section 2.4 describes this in detail.

Expressions whose execution models can be found in neither of the above: Our original estimations apply. For example, specification requires that element constructors copy its content. However, due to a lack of explicit descriptions in formal semantics, and in prior research, for that matter, we have estimated cost based on experiments. The results showed the linear cost in the size of the contents. Section 2.5 elaborates on this subject.

2.3 Cost on evaluation of path expressions

According to [21], data and query complexity (i.e., computational complexity with respect to the size of the input data and input query, respectively) of the path expressions are both polynomial in time. Even linear evaluation time can be assumed for the defined Core XPath there.

We have evaluated Galax[3] [19], which strictly adheres to formal semantics, and another secondary-storage implementation of XQuery called eXist[4] [20], for the behavior of the path expression execution.

[3] `http://www.galaxquery.org/`
[4] `http://exist.sourceforge.net/`

The query used to expand the length of a query size is

$$\texttt{//a/b/}\underbrace{\texttt{parent::a/b/}\ldots\texttt{parent::a/b}}_{|Q|-1}$$

which follows [21]. Similarly, the size of input data $|D|$ is adjusted using data structure denoted in Fig. 2.

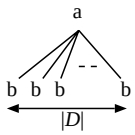

Fig. 2. Data structure used for Gottlob's scalability test

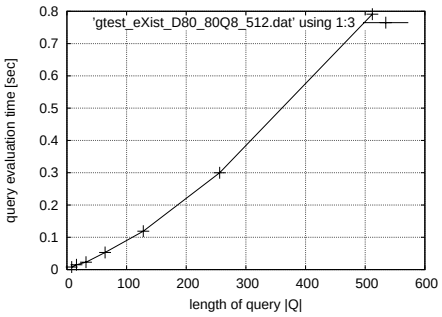

Fig. 3. Gottlob test (eXist [20], $|D| = 80$)

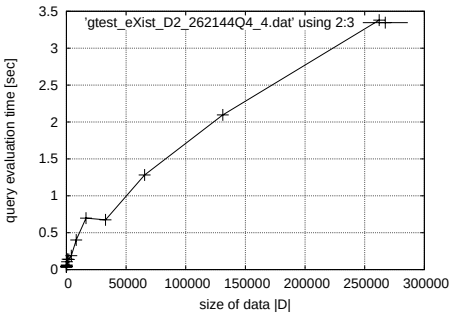

Fig. 4. Gottlob test (eXist, $|Q| = 4$)

eXist behaves linearly with respect to the query size of the test query(Fig. 3). Figure 4 shows the linear behavior with respect to the data size as well.

No evaluation has yet been conducted on the behaviors for more complex data or queries, but Galax proved the exponential behavior (Fig. 5), which implies possible performance degradation in transformations where the number of path expression occurrence is increased.

For Galax, secondary storage implementation using BerkeleyDB[5] is included in the Version 0.4.0 release. However, the exponential behavior seemed to be inherited (Figs. 7 and 8).

[5] `http://www.sleepycat.com/products/db.shtml`

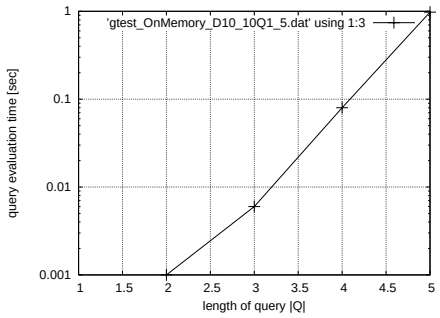

Fig. 5. Gottlob test (Galax 0.4.0 main memory, $|D| = 10$)

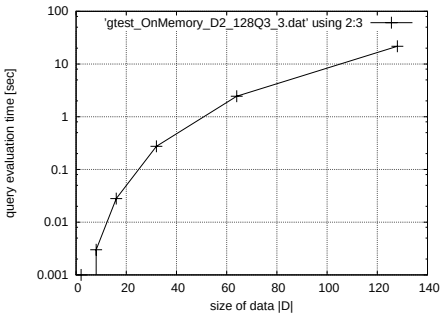

Fig. 6. Gottlob test (Galax 0.4.0 main memory, $|Q| = 3$)

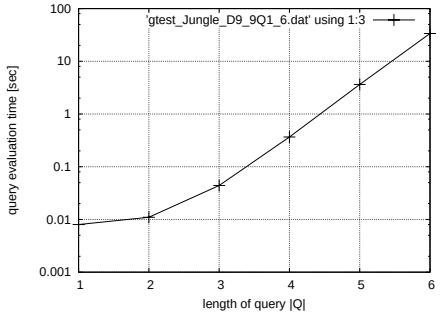

Fig. 7. Gottlob test (Galax 0.4.0 secondary storage, $|D| = 9$)

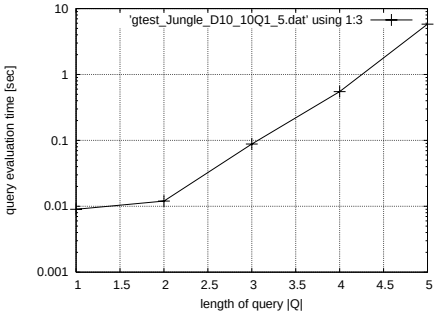

Fig. 8. Gottlob test (Galax 0.4.0 secondary storage, $|D| = 10$)

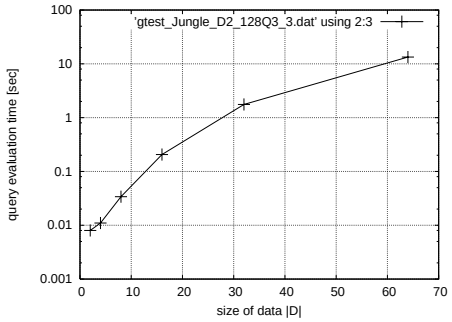

Fig. 9. Gottlob test (Galax 0.4.0 secondary storage, $|Q| = 3$)

Polynomial behavior had been seen with respect to data size $|D|$ for both main-memory and secondary storage implementations (Figs. 6 and 9)[6].

2.4 Cost of monoid comprehension

As FLWOR and quantified expressions correspond to monoid comprehension (accumulator for the FLWOR is a sequence concatenation operator ",", while for quantified expressions it is boolean or/and operators), we refer to [11], which includes the cost model of the comprehension syntax, as in Fig. 10.

$$\text{size}[\![\oplus\{e \parallel \overline{r}\}]\!] = \text{size}[\![\overline{r}]\!] \quad \text{(S1)}$$
$$\text{size}[\![\overline{r}, v \leftarrow e]\!] = \text{size}[\![e]\!] \times \text{size}[\![\overline{r}]\!] \quad \text{(S2)}$$
$$\text{size}[\![\overline{r}, pred]\!] = \text{size}[\![\overline{r}]\!] \times \text{selectivity}[\![pred]\!] \quad \text{(S3)}$$
$$\text{size}[\![\overline{r}, v \equiv e]\!] = \text{size}[\![\overline{r}]\!] \quad \text{(S4)}$$
$$\text{size}[\![]\!] = 1 \quad \text{(S5)}$$

$$\text{cost}[\![\oplus\{e \parallel \overline{r}\}]\!] = \text{cost}[\![e]\!] + \text{cost}[\![\overline{r}]\!] + \text{size}[\![\overline{r}]\!] \quad \text{(C1)}$$
$$\text{cost}[\![\overline{r}, v \leftarrow e]\!] = \text{cost}[\![e]\!] + \text{cost}[\![\overline{r}]\!] \quad \text{(C2)}$$
$$\text{cost}[\![\overline{r}, pred]\!] = \text{cost}[\![pred]\!] + \text{cost}[\![\overline{r}]\!] \quad \text{(C3)}$$
$$\text{cost}[\![\overline{r}, v \equiv e]\!] = \text{cost}[\![e]\!] + \text{cost}[\![\overline{r}]\!] \quad \text{(C4)}$$
$$\text{cost}[\![]\!] = 0 \quad \text{(C5)}$$

Fig. 10. Cost model by [11]

Since both transformation rules and the cost model are provided in the form of a comprehension syntax, they are easily converted to XQuery bidirectionally.

For example, the FLWOR expression

[XQuery] **for** $\$v_1$ **in** e_1
...
let $\$v_2 := e_2$
...
where e_w
return e_r

is equivalent to the comprehension representation

$$,\{e_r \parallel v_1 \leftarrow e_1, \ldots, v_2 \equiv e_2, \ldots, e_w\}$$

whose cost model is quoted in Fig. 10. We follow this style of cost function construction, where three basic functions (cost, size, and selectivity) are combined with each other. However, this model was unable to be directly adopted.

Consider the cost estimation of the following FLWOR expression:

for $\$x$ **in** e_1 **where** *pred* **return** e_3
$\equiv ,\{e_3 \parallel x \leftarrow e_1, pred\}$

[6] Secondary storage implementation ran faster than main-memory implementation for XMark benchmark data and queries.

$$
\begin{aligned}
& \mathrm{cost}[\![, \{e_3 \mid\mid x \leftarrow e_1, \mathit{pred}\}]\!] \\
= \quad & \text{(C1)} \\
& \mathrm{cost}[\![e_3]\!] + \mathrm{cost}[\![x \leftarrow e_1, \mathit{pred}]\!] + \mathrm{size}[\![x \leftarrow e_1, \mathit{pred}]\!] \\
= \quad & \text{(C3),(S3)} \\
& \mathrm{cost}[\![e_3]\!] + \mathrm{cost}[\![x \leftarrow e_1]\!] + \mathrm{cost}[\![\mathit{pred}]\!] + \mathrm{size}[\![x \leftarrow e_1]\!] \times \mathrm{selectivity}[\![\mathit{pred}]\!] \\
= \quad & \text{(C2),(S2)} \\
& \mathrm{cost}[\![e_3]\!] + \mathrm{cost}[\![e_1]\!] + \mathrm{cost}[\![\mathit{pred}]\!] + \mathrm{size}[\![e_1]\!] \times \mathrm{selectivity}[\![\mathit{pred}]\!],
\end{aligned}
$$

which means the cost with respect to e_3 and *pred* do not depend on any other expressions.

The above cost model assumes that the head part, which is equivalent to a **return** clause in FLWOR expression, has to be evaluated only once. On the other hand, the **return** clause is evaluated in proportion to the length of the resultant sequence in the **in** clause after filtering based on specified selectivity. This is not captured in the above cost model. Moreover, *pred*, which is a **where** clause equivalent, is also considered to be evaluated only once. In XQuery, the cost should also be proportional to the sequence size of the **in** expression. Our cost model leads to the following different conclusion:

$$
\begin{aligned}
& \mathrm{cost}[\![\textbf{for } \$x \textbf{ in } e_1 \textbf{ where } \mathit{pred} \textbf{ return } e_3]\!] \\
= \quad & \text{(CWFOR)} \\
& \mathrm{cost}[\![e_1]\!] + \mathrm{size}[\![e_1]\!] \cdot (\mathrm{cost}[\![\mathit{pred}]\!] + \mathrm{selectivity}[\![\mathit{pred}]\!] \cdot \mathrm{cost}[\![e_3]\!]) \\
= \quad & \\
& \mathrm{cost}[\![e_1]\!] + \mathrm{size}[\![e_1]\!] \cdot \mathrm{cost}[\![\mathit{pred}]\!] + \mathrm{size}[\![e_1]\!] \cdot \mathrm{selectivity}[\![\mathit{pred}]\!] \cdot \mathrm{cost}[\![e_3]\!],
\end{aligned}
$$

which successfully captures the cost with respect to the **where** and **return** clauses proportional to the size of the input sequence. Besides, [11] intermixes terms with different dimensions (cost and size) in additive expressions, where ours does not.

2.5 Estimating cost of element construction

Specification requires that an element constructor make a whole copy of its content. However, since a description that could lead to the estimation of the element construction cost itself, was not included in the formal semantics, the cost was not able to be immediately inferred.

However, by intuition, the cost will be proportional to the size of the contents of the element constructor.

In this section, Galax was used to estimate the cost.

Figure 11 shows the topology of the node data that was used in the evaluation. The number of branches (w-ary) and depth (d) was varied to adjust the number of nodes. The relationship between the node number n and w,d is described by the equation

$$
n = \sum_{k=1}^{d-1} w^k = w\frac{w^{d-1} - 1}{w - 1}.
$$

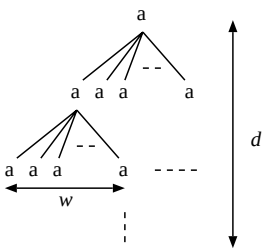

Fig. 11. Topology of trees used in experiments

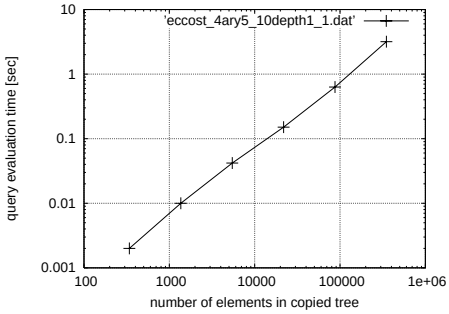

Fig. 12. Scalability analysis of element construction (Galax 0.4.0, one copy)

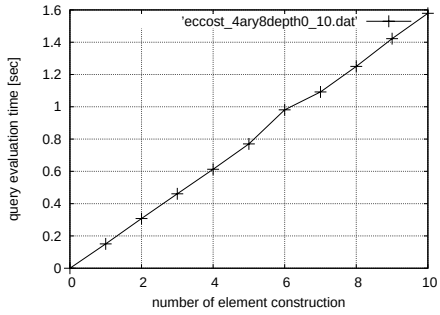

Fig. 13. Scalability analysis of element construction depth=8(Galax 0.4.0)

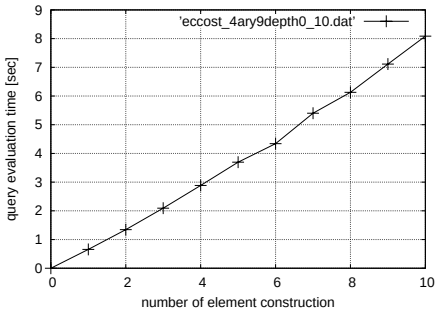

Fig. 14. Scalability analysis of element construction depth=9(Galax 0.4.0)

The query that was used to produce the diverse number n of nodes is shown in Fig. 15.

To exclude time for processing besides than element construction, data that is copied from was loaded from an external XML file rather than generated within the query. The output cost of the copied elements was also excluded by letting Galax return separate constant data instead, as a query result.

The iterated generation of content data could have been implemented using a **for** expression with a range expression as its input. However, to avoid the **for** expression cost getting mixed in, inline direct element constructors were used instead.

In this query, the result of the element construction subexpression is discarded by the projection of following predicate.

If the static analysis had partially evaluated the expression, to discard the redundant element construction, we would not have been able to measure the cost. We confirmed, by examination of the output of the physical plan available in Galax 0.4.0, that no such aggressive optimization had been applied, and consequently the element construction subexpression was actually executed.

```
let $x := $d/a return (1 ,<y>{$x}</y> ... ,<y>{$x}</y> )[1]
                          \_____________n_____________/
```

Fig. 15. Dummy query to measure element construction cost

Element constructors, according to the specification, copy constructs supplied as their arguments and allocates new node IDs for them. This cost is expected to be proportional to the number of nodes and copies. Figure 12 shows the cost with respect to the number of nodes, while Figs. 13 and 14 show the costs with respect to the number of copies. These experiments show the linear behavior of the engines.

3 Cost model

In this section, with reference to the preceding sections, cost models are defined for each language construct (expression) of XQuery. Cost functions are summarized in Table 1.

3.1 Structure of cost model

The cost model consists of the cost function $\mathcal{C}(e)$, which defines the cost of expression e, and two auxiliary functions $\mathcal{S}(e)$ and $\mathcal{P}(e)$.

cost function $\mathcal{C}(e)$ is recursively defined by the cost, size, and probability functions of the subexpressions.

(auxiliary) size function $\mathcal{S}(e)$ is defined in terms of size and probability functions of the subexpressions. It doesn't depend on the cost of the subexpressions.

(auxiliary) probability function $\mathcal{P}(e)$ is defined in terms of the probability and size functions of the subexpressions. It is also independent of the costs of the subexpressions.

3.2 Cost function definitions for each syntax

For expressions

$$\begin{aligned} &\mathcal{C}(\textbf{for } \$x \textbf{ in } s \textbf{ return } f(\$x)) \\ &= \mathcal{C}(s) + \mathcal{S}(s) \cdot \mathcal{C}(f) \end{aligned} \tag{CFOR}$$

In the above XQuery expression and hereafter, an expression that refers to the variables $\$v_1, \$v_2, \ldots, \$v_n$ is denoted by $f(\$v_1, \$v_2, \ldots, \$v_n)$ to clarify its dependence on these variable bindings. Although this syntax is identical to that of a function call, no explicit distinction is made when it is contextually apparent.

Proof. (rationale)

According to dynamic semantics (Dynamic Evaluation section)[7] of **for** expressions

$$\frac{\begin{array}{c} \mathsf{dynEnv} \vdash \mathit{Expr}_1 \Rightarrow \mathit{Item}_1, \ldots, \mathit{Item}_n \\ \mathsf{statEnv} \vdash \mathit{VarRef} \textbf{ of var expands to } \mathit{Variable} \\ \mathsf{dynEnv} + \mathsf{varValue}(\mathit{Variable} \Rightarrow \mathit{Item}_1) \vdash \mathit{Expr}_2 \Rightarrow \mathit{Value}_1 \\ \cdots \\ \mathsf{dynEnv} + \mathsf{varValue}(\mathit{Variable} \Rightarrow \mathit{Item}_n) \vdash \mathit{Expr}_2 \Rightarrow \mathit{Value}_n \end{array}}{\mathsf{dynEnv} \vdash \texttt{for } \mathit{VarRef} \texttt{ in } \mathit{Expr}_1 \texttt{ return } \mathit{Expr}_2 \Rightarrow \mathit{Value}_1, \ldots, \mathit{Value}_n}$$

(costs related to looking up variable symbol tables and variable binding are disregarded here)

$$\begin{aligned} &\mathcal{C}(\mathsf{dynEnv} \vdash \texttt{for } \mathit{VarRef} \texttt{ in } \mathit{Expr}_1 \texttt{ return } \mathit{Expr}_2 \Rightarrow \mathit{Value}_1, \ldots, \mathit{Value}_n) \\ &= \mathcal{C}(\mathsf{dynEnv} + \mathsf{varValue}(\mathit{Variable} \Rightarrow \mathit{Item}_1) \vdash \mathit{Expr}_2 \Rightarrow \mathit{Value}_1) \\ &\quad + \cdots + \\ &\quad \mathcal{C}(\mathsf{dynEnv} + \mathsf{varValue}(\mathit{Variable} \Rightarrow \mathit{Item}_n) \vdash \mathit{Expr}_2 \Rightarrow \mathit{Value}_n) \end{aligned}$$

The number of judgments is equal to n, which is the size of the items in input sequence s. We assume uniform cost for each Expr_i. □

FLWOR(For+Where+Return)

$$\begin{aligned} &\mathcal{C}(\textbf{for } \$x \textbf{ in } s \textbf{ where } p(\$x) \textbf{ return } f(\$x)) \\ &= \mathcal{C}(s) + \mathcal{S}(s) \cdot (\mathcal{C}(p) + \mathcal{P}(p) \cdot \mathcal{C}(f)) \end{aligned} \tag{CWFOR}$$

The cost of the predicate (p) evaluation, as well as its selectivity, is taken into consideration. □

[7] `http://www.w3.org/TR/xquery-semantics/#id-for-expression`

FLWOR with multiple generators
$\mathcal{C}$(**for** \$$x_1$ **in** s_1, \$$x_2$ **in** s_2(\$$x_1$) **return** f(\$$x$))
$= \mathcal{C}(s_1) + \mathcal{S}(s_1) \cdot \mathcal{C}(s_2) + \mathcal{S}(s_1) \cdot \mathcal{S}(s_2) \cdot \mathcal{C}(f)$ □

literals

$\mathcal{C}(literal) =$ constant
For relative cost evaluations, the cost of the literal may be left uninterpreted. □

Direct Element Construction

$$\mathcal{C}(\langle QName \rangle \{e\} \langle / QName \rangle) = \mathcal{C}(e) + \mathcal{S}(e//*) \cdot C_e \qquad \text{(CDEC)}$$

$\mathcal{S}(e//*)$, which is defined at (SDSTEP) later in section 3.3, represents the number of elements copied. C_e denotes the unit cost of the element construction, while the second term, as a whole, captures the deep copy cost of e. □

Computed Element Construction Cost of the computed element construction with literal *QName* is identical to that of the direct element construction, since they are semantically equivalent.

$$\mathcal{C}(\textbf{element } QName \ \{e\}) = \mathcal{C}(e) + \mathcal{S}(e//*) \cdot C_e \qquad \text{(CCEC)}$$

If the *QName* part is not constant, the cost of evaluation of that part is simply added to yield

$$\mathcal{C}(\textbf{element } e_1 \ \{e_2\}) = \mathcal{C}(e_1) + \mathcal{C}(e_2) + \mathcal{S}(e_2//*) \cdot C_e$$

□

Step expression $\mathcal{C}(e/QName)$

if e **is a path expression** then the cost proportional to

$$|D|^c \cdot |Q|^d$$

in [21] is used to yield

$$\mathcal{C}(e/q_1/q_2/\cdots/q_N) = \mathcal{C}(e) + \mathcal{S}(e)^c \times N^d \cdot k \qquad \text{(CGSTEP)}$$

if e **is not a path expression** Substitute N in (CGSTEP) with 1 yields

$$\mathcal{C}(e/QName) = \mathcal{C}(e) + \mathcal{S}(e)^c \cdot k \qquad \text{(CCSTEP)}$$

This is consistent with the following formulation using formal semantics:

left hand side
= normalization of $/QName$ and (2)
$\mathcal{C}(e/\texttt{child::*[self::}QName\texttt{]})$
= (CPRED)
$\mathcal{C}(e/\texttt{child::*}) + \mathcal{S}(e/\texttt{child::*}) \cdot \mathcal{C}(\texttt{self::}QName)$
= normalization rule of path expression
$\mathcal{C}(e) + \mathcal{S}(e) \cdot \mathcal{C}(\texttt{child::*}) + \mathcal{S}(e) \cdot \mathcal{S}(\texttt{child::*}) \cdot \mathcal{C}(\texttt{self::}QName)$
=
$\mathcal{C}(e) + \mathcal{S}(e) \cdot (\mathcal{C}(\texttt{child::*}) + \mathcal{S}(\texttt{child::*}) \cdot \mathcal{C}(\texttt{self::}QName))$
= treat coefficient of $\mathcal{S}(s)$ as constant k
right hand side (with exponent $c = 1$)

□

$\mathcal{C}(e//QName)$

$$\mathcal{C}(e//QName) = \mathcal{C}(e) + k \cdot \mathcal{S}(e//*) \qquad \text{(CDSTEP)}$$

Proof. Using the properties

$$\begin{aligned} & e/\texttt{descendant::node()} \\ = {} & e/\texttt{descendant-or-self::node()/child::node()} && (1) \\ & \mathit{axis}\texttt{::}QName = \mathit{axis}\texttt{::*[self::}QName\texttt{]} && (2) \end{aligned}$$

lhs
= normalization of // and (1)
$\mathcal{C}(e/\texttt{descendant::node()[self::}QName\texttt{]})$
= (CPRED)
$\mathcal{C}(e/\texttt{descendant::node()}) + \mathcal{S}(e/\texttt{descendant::node()}) \cdot \mathcal{C}(\texttt{self::}QName)$
= normalization rule of path expression
$\mathcal{C}(e) + \mathcal{S}(e) \cdot \mathcal{C}(\texttt{descendant::node()})$
$+\mathcal{S}(e/\texttt{descendant::node()}) \cdot \mathcal{C}(\texttt{self::}QName)$
= let k be a unit cost of child axis access
$\mathcal{C}(e) + \mathcal{S}(e) \cdot k \cdot \mathcal{S}(\texttt{descendant::node()})$
$+\mathcal{S}(e/\texttt{descendant::node()}) \cdot \mathcal{C}(\texttt{self::}QName)$
=
$\mathcal{C}(e) + k \cdot \mathcal{S}(e/\texttt{descendant::node()})$
$+\mathcal{S}(e/\texttt{descendant::node()}) \cdot \mathcal{C}(\texttt{self::}QName)$
=
$\mathcal{C}(e) + (k + \mathcal{C}(\texttt{self::}QName)) \cdot \mathcal{S}(e/\texttt{descendant::node()})$
= (1) and let $k' = (k + \mathcal{C}(\texttt{self::}QName))$
rhs

□

Absolute Step expression As the absolute path expressions can be normalized by [10] into relative path expressions, cost functions, such as $\mathcal{C}(//QName)$ can be derived from (CCSTEP):

$$
\begin{aligned}
& \mathcal{C}(/\mathit{QName}) \\
= \quad & \text{normalization rule of } / \\
& \mathcal{C}(\texttt{fn:root(self::node()) treat as document-node()}/\mathit{QName}) \\
= \quad & (\textsc{Ccstep}) \\
& \mathcal{C}(\texttt{fn:root(self::node()) treat as document-node()}) \\
& + \mathcal{S}(\texttt{fn:root(self::node()) treat as document-node()})^c \cdot k \\
= \quad & \\
& \mathcal{C}(\texttt{fn:root(self::node()) treat as document-node()}) + k \quad (\textsc{Cacstep})
\end{aligned}
$$

□

$$
\begin{aligned}
& \mathcal{C}(//\mathit{QName}) \\
= \quad & \text{normalization rule of } // \\
& \mathcal{C}(\texttt{fn:root(self::node()) treat as document-node()}//\mathit{QName}) \\
= \quad & (\textsc{Cdstep}) \\
& \mathcal{C}(\texttt{fn:root(self::node()) treat as document-node()}) \\
& + k \cdot \mathcal{S}(\texttt{fn:root(self::node()) treat as document-node()}//*) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\textsc{Cadstep})
\end{aligned}
$$

□

Comma (sequence construction) expression

$$\mathcal{C}(e_1, e_2) = \mathcal{C}(e_1) + \mathcal{C}(e_2) \qquad (\textsc{Cseq})$$

Cost of concatenation is omitted here. □

Let expression

$$\mathcal{C}(\textbf{let } \$x := s \textbf{ return } f(\$x)) = \mathcal{C}(s) + \mathcal{C}(f) \qquad (\textsc{Clet})$$

Variable reference expression

$$\mathcal{C}(\$x) = \text{constant} = C_{vr} \qquad (\textsc{Cvr})$$

This cost may be ignored depending on the context.

Quantified expression

$$\mathcal{C}(\textbf{some } \$x \textbf{ in } s \textbf{ satisfies } p(\$x)) = \mathcal{C}(s) + \mathcal{S}(s) \cdot \mathcal{C}(p) \qquad \text{(CNSOME)}$$

Note that this formula cannot capture the behavior where the result of a true value is returned as soon as p evaluates to true for some item in input sequence s.

This behavior can be modeled as follows.

$$\begin{aligned}
&\mathcal{C}(s) + \mathcal{C}(p(x_1)) + \mathcal{C}(p(x_2))(1 - \mathcal{P}(p(x_1))) \\
&+ \mathcal{C}(p(x_3))(1 - \mathcal{P}(p(x_1)))(1 - \mathcal{P}(p(x_2))) \\
&\vdots \\
&+ \mathcal{C}(p(x_n))(1 - \mathcal{P}(p(x_1))) \cdots (1 - \mathcal{P}(p(x_{n-1}))) \\
&= \mathcal{C}(s) + \sum_{i=1}^{n} \left(\mathcal{C}(p(x_i)) \prod_{j=1}^{i-1} (1 - \mathcal{P}(p(x_j))) \right)
\end{aligned} \tag{3}$$

where

$$\begin{aligned}
(x_1, x_2, \ldots, x_n) &= s \\
n &= \mathcal{S}(s)
\end{aligned}$$

Moreover, if $\mathcal{C}(p(x_1)) = \mathcal{C}(p(x_2)) = \cdots = \mathcal{C}(p(x_n)) = \mathcal{C}(p)$, $\mathcal{P}(p(x_1)) = \mathcal{P}(p(x_2)) = \cdots = \mathcal{P}(p(x_n)) = \mathcal{P}(p)$ can be assumed as in (CNSOME),

$$\mathcal{C}(s) + \frac{1 - (1 - \mathcal{P}(p))^{\mathcal{S}(s)}}{\mathcal{P}(p)} \mathcal{C}(p) \qquad \text{(CSOME)}$$

Similarly, the cost of universal quantification can be captured by

$$\mathcal{C}(s) + \frac{1 - \mathcal{P}(p)^{\mathcal{S}(s)}}{1 - \mathcal{P}(p)} \mathcal{C}(p) \qquad \text{(CEVERY)}$$

As quantified expressions with multiple variable bindings are equivalent to nested quantified expressions according to normalization rules in the formal semantics, their costs can be calculated by recursive applications of the above cost formulas.

Corollary 1

$$\mathcal{C}(\textbf{some } \$x \textbf{ in } s \textbf{ satisfies } p(\$x)) < \mathcal{C}(s) + \frac{\mathcal{C}(p)}{\mathcal{P}(p)}$$

Figure 16 shows the curve of the cost function with respect to the length of input sequence s, for representative values of p. As the length increases, the cost approaches the right hand side of the inequality. □

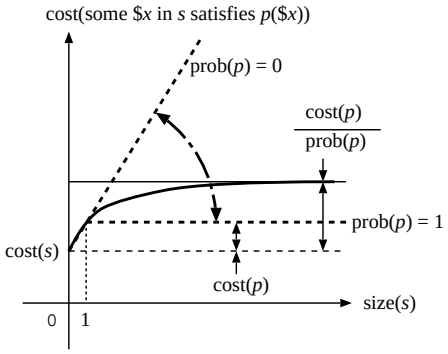

Fig. 16. Cost of quantified expression

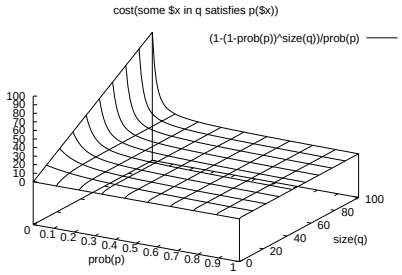

Fig. 17. 3D plot of quantified expression cost

Corollary 2 *For* $\mathcal{P}(p) = 0$*,*

$$\mathcal{C}(\textbf{some } \$x \textbf{ in } s \textbf{ satisfies } p(\$x)) = \mathcal{C}(s) + \mathcal{S}(s) \cdot \mathcal{C}(p)$$

Proof. Substitute $\mathcal{P}(p)$ with 0 in eq. (3). □

$\mathcal{P}(p)$ cannot be replaced by 0 in (CSOME). However, taking limits yields similar results as follows.

Proof.

$$\begin{aligned}
&\text{let } n = \mathcal{S}(s) \\
&\lim_{\mathcal{P}(p) \to +0} \left(\mathcal{C}(s) + \frac{1 - (1 - \mathcal{P}(p))^n}{\mathcal{P}(p)} \mathcal{C}(p) \right) \\
&(\because \tfrac{d}{dx} \{1 - (1-x)^n\} = n(1-x)^{n-1}) \\
= &\lim_{\Delta(\mathcal{P}(p)) \to +0} \left(\mathcal{C}(s) + \frac{n(1 - \Delta\mathcal{P}(p))^{n-1} \Delta\mathcal{P}(p)}{\Delta\mathcal{P}(p)} \mathcal{C}(p) \right) \\
= &\mathcal{C}(s) + n \cdot \mathcal{C}(p)
\end{aligned}$$

□

Conditional expression

$$\mathcal{C}(\textbf{if } (e_1) \textbf{ then } e_2 \textbf{ else } e_3) = \mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(e_2) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(e_3) \quad \text{(CIF)}$$

Logical expression

$$\mathcal{C}(e_1 \textbf{ or } e_2) = \mathcal{C}(e_1) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(e_2) \quad \text{(COR)}$$

$$\mathcal{C}(e_1 \textbf{ and } e_2) = \mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(e_2) \quad \text{(CAND)}$$

The optimized evaluation strategy, in which the result of either operand determines (if possible) the result of the entire expression, without evaluating another operand, is modeled.

Considering that the cost model itself may determine which operand should be tested first, rather than always testing the left operand first, the following formulation models the more efficient evaluation strategies:

$$\mathcal{C}(e_1 \textbf{ or } e_2) = \mathcal{C}(e_l) + (1 - \mathcal{P}(e_l)) \cdot \mathcal{C}(e_u) \qquad \text{(COOR)}$$
$$\mathcal{C}(e_1 \textbf{ and } e_2) = \mathcal{C}(e_l) + \mathcal{P}(e_l) \cdot \mathcal{C}(e_u) \qquad \text{(COAND)}$$

where

$$l = \{j \mid \mathcal{C}(e_j) = \min_{i \in \{1,2\}} \mathcal{C}(e_i)\}$$
$$u = \{j \mid \mathcal{C}(e_j) = \max_{i \in \{1,2\}} \mathcal{C}(e_i)\}$$

There may be implementations that evaluate both operands regardless of the result of either of the operands. In this case, the cost is simply equal to the sum of the costs of both operands. □

Predicate expression

$$\mathcal{C}(e_1[e_2]) = \mathcal{C}(e_1) + \mathcal{S}(e_1) \cdot \mathcal{C}(e_2) \qquad \text{(CPRED)}$$

Function call

$$\begin{aligned} &\mathcal{C}(QName(e_1, e_2, \ldots, e_N)) \\ &= \mathcal{C}(e_1) + \mathcal{C}(e_2) + \cdots + \mathcal{C}(e_N) + \mathcal{C}(\text{function\_body}(QName)) \end{aligned} \qquad \text{(CFCALL)}$$

It does not capture lazy evaluation.

3.3 Definitions of size functions

Built-in function `fn:count()` returns, for argument as a sequence, the number of items. This value is statically estimated following formal semantics:

$$\mathcal{S}(\textbf{for } \$x \textbf{ in } s \textbf{ return } f(\$x)) = \mathcal{S}(s) \cdot \mathcal{S}(f) \qquad \text{(SFOR)}$$

Proof. Estimation is based on dynamic evaluation formal semantics of a **for** expression, similar to the estimation of the cost functions. As

$$\frac{\begin{array}{c} \mathsf{dynEnv} \vdash \mathit{Expr}_1 \Rightarrow \mathit{Item}_1, \ldots, \mathit{Item}_n \\ \mathsf{statEnv} \vdash \mathit{VarRef} \textbf{ of var expands to } \mathit{Variable} \\ \mathsf{dynEnv} + \mathsf{varValue}(\mathit{Variable} \Rightarrow \mathit{Item}_1) \vdash \mathit{Expr}_2 \Rightarrow \mathit{Value}_1 \\ \ldots \\ \mathsf{dynEnv} + \mathsf{varValue}(\mathit{Variable} \Rightarrow \mathit{Item}_n) \vdash \mathit{Expr}_2 \Rightarrow \mathit{Value}_n \end{array}}{\mathsf{dynEnv} \vdash \texttt{for } \mathit{VarRef} \texttt{ in } \mathit{Expr}_1 \texttt{ return } \mathit{Expr}_2 \Rightarrow \mathit{Value}_1, \ldots, \mathit{Value}_n},$$

$\mathcal{S}(\texttt{for}\ \mathit{VarRef}\ \texttt{in}\ \mathit{Expr}_1\ \texttt{return}\ \mathit{Expr}_2)$
$= \mathcal{S}(\mathit{Value}_1\ , \ldots,\ \mathit{Value}_n)$
$= \mathcal{S}(\mathit{Value}_1) + \ldots + \mathcal{S}(\mathit{Value}_n)$
$= \mathcal{S}(\mathit{Expr}_2) + \ldots + \mathcal{S}(\mathit{Expr}_2) = n \cdot \mathcal{S}(\mathit{Expr}_2) = \mathcal{S}(\mathit{Expr}_1) \cdot \mathcal{S}(\mathit{Expr}_2)$.
Note that the size of the result of each **return** expression is considered identical.

□

Similarly,

$$\mathcal{S}(\textbf{for}\ \$x\ \textbf{in}\ s\ \textbf{where}\ p(\$x)\ \textbf{return}\ f(\$x)) = \mathcal{S}(s) \cdot \mathcal{P}(p) \cdot \mathcal{S}(f) \qquad \text{(SWFOR)}$$

□

$\mathcal{S}(\textbf{for}\ \$x_1\ \textbf{in}\ s_1,\ \$x_2\ \textbf{in}\ s_2(\$x_1)\ \textbf{where}\ p(\$x_1, \$x_2)\ \textbf{return}\ f(\$x_1, \$x_2))$
$= \mathcal{S}(s_1) \cdot \mathcal{S}(s_2) \cdot \mathcal{P}(p) \cdot \mathcal{S}(f)$

$$\mathcal{S}(e/\mathit{QName}) = k \cdot \mathcal{P}(\text{nodename} == \mathit{QName}) \cdot \mathcal{S}(e) \qquad \text{(SCSTEP)}$$

$$\mathcal{S}(e//*) = \mathcal{S}(e/\textbf{child}::*) + \sum_{x \in e/\text{child}::*} \mathcal{S}(x//*)$$

$$\mathcal{S}(e//\mathit{QName}) = \mathcal{S}(e//*) \cdot \mathcal{P}(\text{nodename} == \mathit{QName}) \qquad \text{(SDSTEP)}$$

For the absolute path expressions, the same arguments as for (CACSTEP) can apply:

$\mathcal{S}(/\mathit{QName})$
$=$ normalization rule of /
$\mathcal{S}(\texttt{fn:root(self::node()) treat as document-node()}/\mathit{QName})$
$=$ (SCSTEP)
$k \cdot \mathcal{P}(\text{nodename} == \mathit{QName})$
$\cdot\, \mathcal{S}(\texttt{fn:root(self::node()) treat as document-node()})$
$=$
$k \cdot \mathcal{P}(\text{nodename} == \mathit{QName})$ (SACSTEP)

□

$\mathcal{S}(//\mathit{QName})$
$=$ normalization rule of //
$\mathcal{S}(\texttt{fn:root(self::node()) treat as document-node()}//\mathit{QName})$
$=$ (SDSTEP)
$\mathcal{S}(\texttt{fn:root(self::node()) treat as document-node()}//*)$
$\cdot\, \mathcal{P}(\text{nodename} == \mathit{QName})$ (SADSTEP)

□

$\mathcal{S}(literal)$:This value may be left uninterpreted.

$$\mathcal{S}(e_1, e_2) = \mathcal{S}(e_1) + \mathcal{S}(e_2) \quad \text{(SSEQ)}$$
$$\mathcal{S}(\textbf{let } \$x := s \textbf{ return } f(\$x)) = \mathcal{S}(f) \quad \text{(SLET)}$$

$$\mathcal{S}(\$x) = \mathcal{S}(\text{dereference}(\$x)) \quad \text{(SVR)}$$

Note that this is always 1 for the variables $\$x$ bound in **for** or quantified expressions (these are called "*in*-bound" variables in the remainder of this paper).

The size of the expressions that yield scalar (boolean values, for example) is always 1.

$$\mathcal{S}(\textbf{some } \$x \textbf{ in } s \textbf{ satisfies } p(\$x)) = 1 \quad \text{(SSOME)}$$
$$\mathcal{S}(\textbf{every } \$x \textbf{ in } s \textbf{ satisfies } p(\$x)) = 1 \quad \text{(SEVERY)}$$
$$\mathcal{S}(e_1 \textbf{ or } e_2) = 1 \quad \text{(SOR)}$$
$$\mathcal{S}(e_1 \textbf{ and } e_2) = 1 \quad \text{(SAND)}$$
$$\mathcal{S}((e)) = \mathcal{S}(e) \quad \text{(SPAREN)}$$

$$\mathcal{S}(\langle QName \rangle \{e\} \langle / QName \rangle) = 1 \quad \text{(SDEC)}$$
$$\mathcal{S}(\textbf{element } QName \ \{e_1, e_2, \ldots, e_N\}) = 1 \quad \text{(SCEC)}$$

$$\mathcal{S}(\textbf{if } (e_1) \textbf{ then } e_2 \textbf{ else } e_3)$$
$$= \mathcal{P}(e_1) \cdot \mathcal{S}(e_2) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{S}(e_3) \quad \text{(SIF)}$$

$$\mathcal{S}(e_1[e_2]) = \mathcal{P}(e_2) \cdot \mathcal{S}(e_1) \quad \text{(SPRED)}$$

$$\mathcal{S}(QName(e_1, e_2, \ldots, e_N))(\text{function call})$$
$$= \mathcal{S}(\text{function\_body}(QName)) \quad \text{(SFCALL)}$$

More concrete values obtained using [4] may replace the above mentioned size functions where necessary.

3.4 Definitions of probability functions

The probability that boolean expression e evaluates to true is denoted by $\mathcal{P}(e)$, which is defined as follows:

Probability of logical expression yields true

$$\mathcal{P}(e_1 \textbf{ and } e_2) = \mathcal{P}(e_1) \cdot \mathcal{P}(e_2) \tag{PAND}$$

$$\mathcal{P}(e_1 \textbf{ or } e_2) = 1 - (1 - \mathcal{P}(e_1)) \cdot (1 - \mathcal{P}(e_2)) \tag{POR}$$

Probability of quantified expression yields true

$$\begin{aligned} &\mathcal{P}(\textbf{every } \$x \textbf{ in } s \textbf{ satisfies } p(\$x)) \\ &= \mathcal{P}(p)^{\mathcal{S}(q)} \end{aligned} \tag{PEVERY}$$

Probability that a **some** expression is evaluated to true is a complement of the probability that all instances of the body expression are evaluated to false.

$$\begin{aligned} &\mathcal{P}(\textbf{some } \$x \textbf{ in } s \textbf{ satisfies } p(\$x)) \\ &= 1 - (1 - \mathcal{P}(p))^{\mathcal{S}(q)} \end{aligned} \tag{PSOME}$$

4 Cost change estimation for each transformation rule using proposed cost model

4.1 Role of cost change calculation

In this section, the cost change for various transformations applied for the purpose of optimization is proved using the cost functions defined in this paper. These proofs inductively imply that a program transformation that consists only of transformations that maintain or reduce costs will always maintain or improve costs.

Semantic equivalence of these transformations is proved in appendix A.

4.2 Rules and cost changes

Theorem 1 (monad left unit law (for expression)). *Transformation*

$$\frac{\mathit{is\_singleton}(s)}{\textbf{for } \$x \textbf{ in } s \textbf{ return } f(\$x) \Rightarrow \textbf{let } \$x := s \textbf{ return } f(\$x)} \tag{TFLU}$$

doesn't change cost.

Proof.

$$\begin{aligned} &\mathcal{C}(\text{lhs}) \\ =\ &(\text{CFOR}) \\ &\mathcal{C}(s) + \mathcal{S}(s) \cdot \mathcal{C}(f) \end{aligned} \qquad \begin{aligned} &\mathcal{C}(\text{rhs}) \\ =\ &(\text{CLET}) \\ &\mathcal{C}(s) + \mathcal{C}(f) \end{aligned}$$

By the way, $\mathit{is\_singleton}(s)$ implies $\mathcal{S}(s) = 1$. Thus, the costs of both sides are equal. □

Theorem 2 (monad left unit law (quantified expressions)). *Transformation*

$$\frac{\mathit{is\_singleton}(s)}{\textbf{some } \$x \textbf{ in } s \textbf{ satisfies } f(\$x) \Rightarrow \textbf{let } \$x := s \textbf{ return } f(\$x)} \quad (\textsc{Tslu})$$

doesn't change cost.

Proof.

$$\begin{array}{l} \mathcal{C}(\text{lhs}) \\ = \quad \text{by } (\textsc{Cnsome}) \\ \mathcal{C}(s) + \mathcal{S}(s) \cdot \mathcal{C}(f) \end{array} \qquad \begin{array}{l} \mathcal{C}(\text{lhs}) \\ = \quad \text{by } (\textsc{Csome}) \\ \mathcal{C}(s) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(s)}}{\mathcal{P}(f)} \cdot \mathcal{C}(f) \end{array} \qquad \begin{array}{l} \mathcal{C}(\text{rhs}) \\ = \quad (\textsc{Clet}) \\ \mathcal{C}(s) + \mathcal{C}(f) \end{array}$$

By the way, the premise implies $\mathcal{S}(s) = 1$. Therefore, both cost definitions (Cnsome),(Csome) lead to identical costs for both sides, yielding the same conclusion for the **for** version. The same arguments apply for universally quantified (**every**) expressions.

$$\frac{\mathit{is\_singleton}(s)}{\textbf{every } \$x \textbf{ in } s \textbf{ satisfies } f(\$x) \Rightarrow \textbf{let } \$x := s \textbf{ return } f(\$x)} \quad (\textsc{Telu})$$

□

Theorem 3 (monad right unit law (for expression)). *Transformation*

$$\textbf{for } \$x \textbf{ in } s \textbf{ return } \$x = s \quad (\textsc{Tfru})$$

reduces costs.

Proof.

$$\begin{array}{l} \mathcal{C}(\text{lhs}) \\ = \quad (\textsc{Cfor}) \\ \mathcal{C}(s) + \mathcal{S}(s) \cdot \mathcal{C}(\$x) \end{array} \qquad \begin{array}{l} \mathcal{C}(\text{rhs}) \\ = \\ \mathcal{C}(s) \end{array}$$

Thus, $\mathcal{C}(\text{lhs}) > \mathcal{C}(\text{rhs})$. □

Disregarding variable reference cost leads to cost preservation, but not an increase.

Corollary For path expression s:
$\mathcal{C}(lhs) = \mathcal{C}(\textbf{for } \$x \textbf{ in } e/\text{a } \textbf{return } \$x)$
$= \mathcal{C}(e/\text{a}) + \mathcal{S}(e/\text{a}) \cdot \mathcal{C}(\$x) = \mathcal{C}(e/\text{a}) + \mathcal{S}(e/\text{a}) \cdot C_{vr}$
$\mathcal{C}(\text{rhs}) = \mathcal{C}(e/\text{a})$
Hence, the difference of both sides does not depend on the difference in the cost functions (Whether Gottlob's evaluation strategy is used or not doesn't affect the difference).

Monad right unit law (quantified expression) Unlike **for** expressions, quantified expressions with identity (variable reference only) as their bodies do not yield identities. This can be explained in terms of Φ_Q[8] as follows.

$$\begin{aligned} & \textbf{some } \$x \textbf{ in } s \textbf{ satisfies } \$x \\ = \; & \Phi_Q \ (\lambda x.x) \ \vee \ \text{False } s \\ = \; & s_1 \vee s_2 \vee \cdots \vee s_N \end{aligned}$$

The difference comes from the fact that the input and the output monoids are different(((), ",") and (False, $\vee$)). Therefore, we do not deal with such transformations.

Theorem 4 (monad associative law (for expression)). *Transformation*

$$\begin{array}{l} \textbf{for } \$x \textbf{ in for } \$y \textbf{ in } q \\ \qquad\qquad \textbf{return } g(\$y) \\ \textbf{return } f(\$x) \end{array} \quad = \quad \begin{array}{l} \textbf{for } \$y \textbf{ in } q \\ \textbf{return for } \$x \textbf{ in } g(\$y) \\ \qquad\qquad \textbf{return } f(\$x) \end{array} \qquad \text{(TFA)}$$

doesn't change cost.

Proof.

$$\begin{array}{ll} & \mathcal{C}(\text{lhs}) \\ = & \quad \text{(CFOR)} \\ & \mathcal{C}(\textbf{for } \$y \textbf{ in } q \textbf{ return } g(\$y)) + \mathcal{S}(\textbf{for } \$y \textbf{ in } q \textbf{ return } g(\$y)) \cdot \mathcal{C}(f(\$x)) \\ = & \quad \text{(CFOR),(SFOR)} \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(g(\$y)) + \mathcal{S}(g(\$y)) \cdot \mathcal{S}(q) \cdot \mathcal{C}(f(\$x)) \end{array}$$

$$\begin{array}{ll} & \mathcal{C}(\text{rhs}) \\ = & \quad \text{(CFOR)} \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(\textbf{for } \$x \textbf{ in } g(\$y) \textbf{ return } f(\$x)) \\ = & \quad \text{(CFOR)} \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(g(\$y)) + \mathcal{S}(g(\$y)) \cdot \mathcal{C}(f(\$x))) \\ = & \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(g(\$y)) + \mathcal{S}(g(\$y)) \cdot \mathcal{S}(q) \cdot \mathcal{C}(f(\$x)) \end{array}$$

Therefore, the cost of both sides are equal. □

Theorem 5 (associative law of for expressions that has where clauses). *Transformation*

$$\begin{array}{l} \textbf{for } \$x \textbf{ in for } \$y \textbf{ in } q \\ \qquad\qquad \textbf{where } h(\$y) \\ \qquad\qquad \textbf{return } g(\$y) \\ \textbf{where } e(\$x) \\ \textbf{return } f(\$x) \end{array} \quad = \quad \begin{array}{l} \textbf{for } \$y \textbf{ in } q \\ \textbf{where } h(\$y) \\ \textbf{return for } \$x \textbf{ in } g(\$y) \\ \qquad\qquad \textbf{where } e(\$x) \\ \qquad\qquad \textbf{return } f(\$x) \end{array} \qquad \text{(TWFA)}$$

doesn't change cost.

[8] Φ_Q is defined in appendix A.

Proof.

$$
\begin{array}{ll}
& \mathcal{C}(\text{lhs}) \\
= & \quad (\textsc{Cwfor}) \\
& \mathcal{C}(\textbf{for } \$y \textbf{ in } q \textbf{ where } h(\$y) \textbf{ return } g(\$y)) \\
& +\mathcal{S}(\textbf{for } \$y \textbf{ in } q \textbf{ where } h(\$y) \textbf{ return } g(\$y)) \cdot (\mathcal{C}(e) + \mathcal{P}(e) \cdot \mathcal{C}(f)) \\
= & \quad (\textsc{Cwfor}),(\textsc{Swfor}) \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(h) + \mathcal{S}(q) \cdot \mathcal{P}(h) \cdot \mathcal{C}(g) + \mathcal{S}(q) \cdot \mathcal{P}(h) \cdot \mathcal{S}(g) \cdot \mathcal{C}(e) \\
& +\mathcal{S}(q) \cdot \mathcal{P}(h) \cdot \mathcal{S}(g) \cdot \mathcal{P}(e) \cdot \mathcal{C}(f)
\end{array}
$$

$$
\begin{array}{ll}
& \mathcal{C}(\text{rhs}) \\
= & \quad (\textsc{Cwfor}) \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(h) + \mathcal{P}(h) \cdot \mathcal{C}(\textbf{for } \$x \textbf{ in } g(\$y) \textbf{ where } e(\$x) \textbf{ return } f(\$x)) \\
= & \quad (\textsc{Cwfor}) \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(h) + \mathcal{P}(h) \cdot (\mathcal{C}(g) + \mathcal{S}(g) \cdot (\mathcal{C}(e) + \mathcal{P}(e) \cdot \mathcal{C}(f)))) \\
= & \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(h) + \mathcal{S}(q) \cdot \mathcal{P}(h) \cdot (\mathcal{C}(g) + \mathcal{S}(g) \cdot (\mathcal{C}(e) + \mathcal{P}(e) \cdot \mathcal{C}(f))) \\
= & \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(h) + \mathcal{S}(q) \cdot \mathcal{P}(h) \cdot \mathcal{C}(g) + \mathcal{S}(q) \cdot \mathcal{P}(h) \cdot \mathcal{S}(g) \cdot \mathcal{C}(e) \\
& +\mathcal{S}(q) \cdot \mathcal{P}(h) \cdot \mathcal{S}(g) \cdot \mathcal{P}(e) \cdot \mathcal{C}(f)
\end{array}
$$

Thus, the cost of both sides are equal. □

Theorem 6 (associative law for quantified expression). *Transformations*

$$
\begin{array}{l}
\textbf{some } \$x \textbf{ in } (\textbf{for } \$y \textbf{ in } q \textbf{ return } g(\$y)) \\
\textbf{satisfies } f(\$x)
\end{array}
\quad = \quad
\begin{array}{l}
\textbf{some } \$y \textbf{ in } q \\
\textbf{satisfies some } \$x \textbf{ in } g(\$y) \\
\qquad\qquad \textbf{satisfies } f(\$x)
\end{array}
\qquad (\textsc{Tsa})
$$

doesn't change or even improves cost. For universal quantification,

$$
\begin{array}{l}
\textbf{every } \$x \textbf{ in } (\textbf{for } \$y \textbf{ in } q \textbf{ return } g(\$y)) \\
\textbf{satisfies } f(\$x)
\end{array}
\quad = \quad
\begin{array}{l}
\textbf{every } \$y \textbf{ in } q \\
\textbf{satisfies every } \$x \textbf{ in } g(\$y) \\
\qquad\qquad \textbf{satisfies } f(\$x)
\end{array}
\qquad (\textsc{Tea})
$$

Proof using cost(Cnsome)

Proof.

$$
\begin{array}{ll}
& \mathcal{C}(\text{lhs}) \\
= & \quad (\textsc{Cnsome}) \\
& \mathcal{C}(\textbf{for } \$y \textbf{ in } q \textbf{ return } g(\$y)) + \mathcal{S}(\textbf{for`} \$y \textbf{ in } q \textbf{ return } g(\$y)) \cdot \mathcal{C}(f(\$x)) \\
= & \quad (\textsc{Cfor}),(\textsc{Sfor}) \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(g) + \mathcal{S}(g) \cdot \mathcal{S}(q) \cdot \mathcal{C}(f)
\end{array}
$$

$$
\begin{array}{ll}
& \mathcal{C}(\text{rhs}) \\
= & \quad (\textsc{Cnsome}) \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(\textbf{some } \$x \textbf{ in } g(\$y) \textbf{ satisfies } f(\$x)) \\
= & \quad (\textsc{Cnsome}) \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(g) + \mathcal{S}(g) \cdot \mathcal{C}(f)) \\
= & \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(g) + \mathcal{S}(g) \cdot \mathcal{S}(q) \cdot \mathcal{C}(f)
\end{array}
$$

Therefore, the costs of both sides are equal. □

Similar arguments apply for universal quantification.

Proof using cost (Csome)

Proof.

$$
\begin{array}{ll}
& \mathcal{C}(\text{lhs}) \\
= & \quad (\textsc{Csome}) \\
& \mathcal{C}(\textbf{for } \$y \textbf{ in } q \textbf{ return } g(\$y)) + \frac{\mathcal{C}(f) \cdot (1-(1-\mathcal{P}(f))^{\mathcal{S}(\textbf{for } \$y \textbf{ in } q \textbf{ return } g(\$y))})}{\mathcal{P}(f)} \\
= & \quad (\textsc{Cfor}),(\textsc{Sfor}) \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(g) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(q)\cdot\mathcal{S}(g)}}{\mathcal{P}(f)} \cdot \mathcal{C}(f)
\end{array}
$$

$$
\begin{array}{ll}
& \mathcal{C}(\text{rhs}) \\
= & \quad (\textsc{Csome}) \\
& \mathcal{C}(q) + \frac{\mathcal{C}(\textbf{some } \$x \textbf{ in } g(\$y) \textbf{ satisfies } f(\$x))}{\mathcal{P}(\textbf{some } \$x \textbf{ in } g(\$y) \textbf{ satisfies } f(\$x))} \\
& \cdot (1 - (1 - \mathcal{P}(\textbf{some } \$x \textbf{ in } g(\$y) \textbf{ satisfies } f(\$x)))^{\mathcal{S}(q)}) \\
= & \quad (\textsc{Csome}),(\textsc{Psome}) \\
& \mathcal{C}(q) + \frac{\mathcal{C}(g) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}}{\mathcal{P}(f)} \cdot \mathcal{C}(f)}{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}} \cdot (1 - (1 - (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}) \\
= & \\
& \mathcal{C}(q) + \frac{\mathcal{C}(g) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}}{\mathcal{P}(f)} \cdot \mathcal{C}(f)}{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}} \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)\cdot\mathcal{S}(q)}) \\
= & \\
& \mathcal{C}(q) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)\cdot\mathcal{S}(q)}}{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}} \cdot \mathcal{C}(g) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)\cdot\mathcal{S}(q)}}{\mathcal{P}(f)} \cdot \mathcal{C}(f)
\end{array}
$$

By definition of the cost model of **for** expressions, uniform size/cost of $g(\$y)$ is assumed, and by definition of the cost model of **some** expressions, the expected value of $f(\$x)$ is assumed to be uniform.

By the way, as $\mathcal{P}(f)$ approaches 0, the predicate of the **some** expressions has to be evaluated for every item of the input sequences; chances of finding cases where f evaluates to true become smaller, and consequently, the cost difference before and after transformation approaches 0 as well.

The cost difference only appears at the second term, so

$$
\begin{array}{l}
\mathcal{C}(\text{lhs}) - \mathcal{C}(\text{rhs}) \\
= \left(\mathcal{S}(q) - \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)\cdot\mathcal{S}(q)}}{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}}\right) \cdot \mathcal{C}(g).
\end{array}
$$

As $\mathcal{P}(f)$ approaches 0 in the above expression, the second term of the coefficient of $\mathcal{C}(g)$ approaches $\mathcal{S}(q)$.

$(1-\mathcal{P}(f))^{\mathcal{S}(g)}$ represents the probability that f yields false $\mathcal{S}(g)$ times. The second term, namely

$$\left(\sum_{k=0}^{\mathcal{S}(q)-1}((1-\mathcal{P}(f))^{\mathcal{S}(g)})^k\right)\cdot\mathcal{C}(g),$$

represents that for the inner **some** expression, only if for every binding of $\$y$, the previous evaluation of f always yields false for every item of $g(\$y)$, it is evaluated for the next binding of $\$y$.

Furthermore, since this coefficient of $\mathcal{C}(g(\$y))$ is always smaller than $\mathcal{S}(q)$, the cost is always improved by the transformation. □

Next, we would like to study cases where **for** expressions on the lhs include **where** clauses. The corresponding associative law can be defined as,

some $\$y$ **in** (**for** $\$z$ **in** q **where** $h(\$z)$ **return** $g(\$z)$)
satisfies $f(\$y)$
=
some $\$z$ **in** q
satisfies
  ($h(\$z)$ **and** (**some** $\$y$ **in** $g(\$z)$ **satisfies** $f(\$y)$)),

and the cost reduction by this transformation can also be proved.

Proof. Let $p(\$z) \stackrel{\text{def}}{=} h(\$z)$ **and** (**some** $\$y$ **in** $g(\$z)$ **satisfies** $f(\$y)$).

$$\begin{aligned}
&\mathcal{P}(p(\$z))\\
=\ &\text{(PAND)}\\
&\mathcal{P}(h)\cdot\mathcal{P}(\textbf{some } \$y \textbf{ in } g(\$z) \textbf{ satisfies } f(\$y))\\
=\ &\text{(PSOME)}\\
&\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)})
\end{aligned}$$

$$\begin{aligned}
&\mathcal{C}(\text{lhs})\\
=\ &\text{(CSOME)}\\
&\mathcal{C}(\textbf{for } \$z \textbf{ in } q \textbf{ where } h(\$z) \textbf{ return } g(\$z))\\
&+\mathcal{C}(f)\cdot\tfrac{1-(1-\mathcal{P}(f))^{\mathcal{S}(\textbf{for } \$z \textbf{ in } q \textbf{ where } h(\$z) \textbf{ return } g(\$z))}}{\mathcal{P}(f)}\\
=\ &\text{(CFOR),(SFOR)}\\
&\mathcal{C}(q)+\mathcal{S}(q)\cdot(\mathcal{C}(h)+\mathcal{P}(h)\cdot\mathcal{C}(g))\\
&+\mathcal{C}(f)\cdot\tfrac{1-(1-\mathcal{P}(f))^{\mathcal{S}(q)\cdot\mathcal{P}(h)\cdot\mathcal{S}(g(\$z))}}{\mathcal{P}(f)}\\
=\ &\\
&\mathcal{C}(q)+\mathcal{S}(q)\cdot(\mathcal{C}(h)+\mathcal{P}(h)\cdot\mathcal{C}(g))+\tfrac{\mathcal{C}(f)}{\mathcal{P}(f)}\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(q)\cdot\mathcal{P}(h)\cdot\mathcal{S}(g)})\\
=\ &\\
&\mathcal{C}(q)+\mathcal{S}(q)\cdot\mathcal{C}(h)+\mathcal{P}(h)\cdot\mathcal{S}(q)\cdot\mathcal{C}(g)+\tfrac{1-(1-\mathcal{P}(f))^{\mathcal{S}(q)\cdot\mathcal{P}(h)\cdot\mathcal{S}(g)}}{\mathcal{P}(f)}\cdot\mathcal{C}(f)
\end{aligned}$$

$$
\begin{aligned}
& \mathcal{C}(\text{rhs}) \\
= \quad & (\textsc{Csome}) \\
& \mathcal{C}(q) + \tfrac{\mathcal{C}(p)}{\mathcal{P}(p)} \cdot (1 - (1 - \mathcal{P}(p))^{\mathcal{S}(q)}) \\
= \quad & (\textsc{Cand}) \\
& \mathcal{C}(q) + \tfrac{\mathcal{C}(h) + \mathcal{P}(h) \cdot \mathcal{C}(\textbf{some } \$y \textbf{ in } g(\$z) \textbf{ satisfies } f(\$y))}{\mathcal{P}(p)} \cdot (1 - (1 - \mathcal{P}(p))^{\mathcal{S}(q)}) \\
= \quad & (\textsc{Csome}),(\textsc{Pand}) \\
& \mathcal{C}(q) + \tfrac{\mathcal{C}(h) + \mathcal{P}(h) \cdot (\mathcal{C}(g) + \frac{\mathcal{C}(f)}{\mathcal{P}(f)} \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}))}{\mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)})} \\
& \cdot (1 - (1 - \mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}) \\
= \quad & \\
& \mathcal{C}(q) \\
& + (\mathcal{C}(h) + \mathcal{P}(h) \cdot \mathcal{C}(g) + \tfrac{\mathcal{P}(h) \cdot \mathcal{C}(f)}{\mathcal{P}(f)} \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)})) \\
& \cdot \tfrac{1 - (1 - \mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}}{\mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)})} \\
= \quad & \\
& \mathcal{C}(q) + \tfrac{1 - (1 - \mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}}{\mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)})} \cdot \mathcal{C}(h) \\
& + \tfrac{1 - (1 - \mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}}{1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}} \cdot \mathcal{C}(g) + \tfrac{1 - (1 - \mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}}{\mathcal{P}(f)} \cdot \mathcal{C}(f)
\end{aligned}
$$

For each term in the above cost formulas, the cost is proven to be maintained or reduced.

For costs related to $\mathcal{C}(h)$, h is always evaluated $\mathcal{S}(q)$ times on the lhs, while for the rhs, the evaluation of the outermost **some** expression can be stopped as soon as the predicate evaluates to true. Therefore, the cost is always improved by the transformation.

Proof. $\mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)})$ denotes the probability that h is true and f evaluates to false $\mathcal{S}(g)$ times, so the first factor of the first term

$$\left(\sum_{k=0}^{\mathcal{S}(q)-1} (1 - \mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}))^k \right) \cdot \mathcal{C}(h),$$

represents the expected number of evaluations of h.

As $(1 - \mathcal{P}(h) \cdot (1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}))$ denotes the probability that the outer **satisfies** clause evaluates to false, its expected value is always smaller than $\mathcal{S}(q)$. Therefore, rhs is cheaper than lhs with respect to $\mathcal{C}(h)$. □

For the cost regarding the evaluation of g, transformation always improves the cost.

Proof. On the rhs, the coefficient of $\mathcal{C}(g)$ equals to the coefficient of $\mathcal{C}(h)$ multiplied by $\mathcal{P}(h)$. On the lhs, the coefficient is $\mathcal{S}(q) \cdot \mathcal{P}(h)$. Preceding proof showed that the coefficient of $\mathcal{C}(h)$ is smaller than $\mathcal{S}(q)$. Therefore, even if they are multiplied by the same value $\mathcal{P}(h)$, rhs is always smaller than lhs. □

Lastly, cost with respect to the evaluation of f is always improved by the transformation.

Proof. Cost difference (lhs−rhs) with respect to the cost regarding the evaluation of f is

$$\frac{((1-(1-\mathcal{P}(f))^{\mathcal{S}(q)\cdot\mathcal{P}(h)\cdot\mathcal{S}(g)})-(1-(1-\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}))}{\mathcal{P}(f)}\cdot\mathcal{C}(f)$$

$$=$$

$$((((1-\mathcal{P}(f))^{\mathcal{P}(h)})^{\mathcal{S}(g)}))^{\mathcal{S}(q)}-(1-\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)})\cdot\frac{\mathcal{C}(f)}{\mathcal{P}(f)}$$

Let $(1-\mathcal{P}(f))^{\mathcal{S}(g)}=y$. Then the above expression looks like

$$((1-\mathcal{P}(h)\cdot(1-y))^{\mathcal{S}(q)}-(y^{\mathcal{P}(h)})^{\mathcal{S}(q)})\frac{\mathcal{C}(f)}{\mathcal{P}(f)}.$$

We would like to show that $(1-\mathcal{P}(h)\cdot(1-y))^{\mathcal{S}(q)}-(y^{\mathcal{P}(h)})^{\mathcal{S}(q)}$ is always positive, while paying attention to the fact that $0\leq\mathcal{P}(h)\leq 1$ and $0<y\leq 1$.

As both terms have equal exponents, only the base numbers are compared to determine their inequality relationship.

When $y=1$, both terms are equal to 1.

As

$$\frac{d}{dy}(\text{1st term})=\mathcal{P}(h),\ \frac{d}{dy}(\text{2nd term})=\mathcal{P}(h)\cdot y^{\mathcal{P}(h)-1},$$

$$\frac{d}{dy}(\text{2nd term})\bigg|_{y=1}=\mathcal{P}(h)$$

As $\mathcal{P}(h)$ is less than 1, this differential coefficient monotonically increases as y decreases. Therefore, it is always the case where 2nd term $\leq$ 1st term. Thus, the cost on the rhs, with respect to evaluation of f, is shown to always be less than that on the rhs. □

Therefore, all the coefficients of $\mathcal{C}(h)$,$\mathcal{C}(g)$,$\mathcal{C}(f)$, excluding $\mathcal{C}(q)$, are shown to decrease by the transformation.

Consequently, the total cost, which is the sum of these costs, is always reduced. □

Effect of the difference in evaluation strategy of and expression to the associative law with where clause Despite the above proofs, we have observed a cost increase in the application of the associative law for Galax 0.3.5 and eXist snapshot-20041119.

We have concluded that the evaluation strategy of **and** expressions in both implementations, where both operands are always evaluated(in Galax 0.4.0, the right operand is not evaluated if the left operand is evaluated to be false) accounts for the speed reduction.

Actually, we can model the speed reduction by incorporating the evaluation strategy of Galax 0.3.5 into our cost model in the following way.

For $p(\$z) \stackrel{\text{def}}{=} h(\$z)$ **and** (**some** $\$y$ **in** $g(\$z)$ **satisfies** $f(\$y)$), cost is modeled by

$$\begin{aligned}
&\mathcal{C}(p(\$z)) \\
&= \mathcal{C}(h) + \mathcal{C}(\textbf{some } \$y \textbf{ in } g(\$z) \textbf{ satisfies } f(\$y)) \\
&= \mathcal{C}(h) + \mathcal{C}(g) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}}{\mathcal{P}(f)}\mathcal{C}(f)
\end{aligned}$$

Note that the cost of both operands for the **and** expression are simply added.

$$\begin{aligned}
&\mathcal{C}(\text{rhs}) \\
= \quad &(\textsc{Csome}) \\
&\mathcal{C}(q) + \frac{1-(1-\mathcal{P}(p))^{\mathcal{S}(q)}}{\mathcal{P}(p)} \cdot \mathcal{C}(p) \\
= \quad &\text{definition of } p \text{ above and } (\textsc{Pand}) \\
&\mathcal{C}(q) + \frac{1-(1-\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}}{\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)})} \\
&\cdot(\mathcal{C}(h) + \mathcal{C}(g) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}}{\mathcal{P}(f)}\mathcal{C}(f)) \\
= \quad & \\
&\mathcal{C}(q) + \frac{1-(1-\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}}{\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)})} \cdot \mathcal{C}(h) \\
&+\frac{1-(1-\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}}{\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)})} \cdot \mathcal{C}(g) \\
&+\frac{1-(1-\mathcal{P}(h)\cdot(1-(1-\mathcal{P}(f))^{\mathcal{S}(g)}))^{\mathcal{S}(q)}}{\mathcal{P}(h)\cdot\mathcal{P}(f)} \cdot \mathcal{C}(f)
\end{aligned}$$

Compared to the optimized evaluation of **and** expressions, the cost with respect to the evaluation of g and f is changed. More specifically, they are multiplied by $1/\mathcal{P}(h)$. As $\mathcal{P}(h)$ is smaller than 1, it causes a cost increase. This increment becomes larger for smaller $\mathcal{P}(h)$.

Furthermore, depending on the value of $\mathcal{P}(h)$, the cost of rhs may be larger than that of lhs.

As $\mathcal{P}(h)$ becomes smaller, chances of g being evaluated becomes very limited. So, the cost of a **for** expression approaches $\mathcal{C}(q) + \mathcal{C}(h)\cdot\mathcal{S}(q)$, and the size of a **for** expression approaches 0. Lhs as a whole approaches

$$\mathcal{C}(q) + \mathcal{S}(q)\cdot\mathcal{C}(h) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{P}(h)\cdot\mathcal{S}(q)\cdot\mathcal{S}(g)}}{\mathcal{P}(f)} \cdot \mathcal{C}(f)$$

and the third term approaches $\mathcal{P}(h)\cdot\mathcal{S}(q)\cdot\mathcal{S}(g)\cdot\mathcal{C}(f)$ as $\mathcal{P}(f)$ becomes smaller.

On the rhs, the number of evaluations of the outer **satisfies** clause approaches $\mathcal{S}(q)$. As the inner **some** expression is evaluated as many times as $h(\$z)$ is evaluated, the cost with respect to $g(\$z)$ approaches $\mathcal{S}(q)\cdot\mathcal{C}(g)$.

Therefore, the cost of rhs approaches

$$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(h) + \mathcal{S}(q) \cdot \mathcal{C}(g) + \mathcal{S}(q) \cdot \frac{1 - (1 - \mathcal{P}(f))^{\mathcal{S}(g)}}{\mathcal{P}(f)} \mathcal{C}(f) \ .$$

It can be concluded that the rhs becomes heavier as the selection by the **where** clause no longer works for cost regarding to g.

For implementations that always evaluate both operands of the **and** expression, the associative law can be expressed in terms of **if** expressions to avoid this cost increase. As it is semantically equivalent to the **and** version, this will be the implementation-independent associative law.

$$\begin{array}{l} \textbf{some } \$y \textbf{ in } (\textbf{for } \$z \textbf{ in } q \\ \qquad\qquad \textbf{where } h(\$z) \\ \qquad\qquad \textbf{return } g(\$z)) \\ \textbf{satisfies } f(\$y) \end{array} = \begin{array}{l} \textbf{some } \$z \textbf{ in } q \\ \textbf{satisfies if } (h(\$z)) \\ \qquad\qquad \textbf{then some } \$y \textbf{ in } g(\$z) \\ \qquad\qquad\qquad \textbf{satisfies } f(\$y) \\ \qquad\qquad \textbf{else } \texttt{fn:false()} \end{array} \qquad \text{(TSAIF)}$$

Theorem 7 (distribution (for expression)). *Transformation*

$$\begin{array}{l} \textbf{for } \$x \textbf{ in } (q,\ q') \\ \textbf{return } f(\$x) \end{array} = \begin{array}{l} (\textbf{for } \$x \textbf{ in } q \textbf{ return } f(\$x)), \\ (\textbf{for } \$x \textbf{ in } q' \textbf{ return } f(\$x)) \end{array} \qquad \text{(TFD)}$$

doesn't change cost.

Proof.

$$\begin{array}{ll} & \mathcal{C}(\text{lhs}) \\ = & \quad \text{(CFOR)} \\ & \mathcal{C}(q,\ q') + \mathcal{S}(q,\ q') \cdot \mathcal{C}(f(\$x)) \\ = & \quad \text{(CSEQ),(SSEQ)} \\ & \mathcal{C}(q) + \mathcal{C}(q') + (\mathcal{S}(q) + \mathcal{S}(q')) \cdot \mathcal{C}(f(\$x)) \\ = & \\ & \mathcal{C}(q) + \mathcal{C}(q') + \mathcal{S}(q) \cdot \mathcal{C}(f(\$x)) + \mathcal{S}(q') \cdot \mathcal{C}(f(\$x)) \end{array}$$

$$\begin{array}{ll} & \mathcal{C}(\text{rhs}) \\ = & \quad \text{(CSEQ)} \\ & \mathcal{C}(\textbf{for } \$x \textbf{ in } q \textbf{ return } f(\$x)) + \mathcal{C}(\textbf{for } \$x \textbf{ in } q' \textbf{ return } f(\$x)) \\ = & \quad \text{(CFOR)} \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(f(\$x)) + \mathcal{C}(q') + \mathcal{S}(q') \cdot \mathcal{C}(f(\$x)) \end{array}$$

Hence, $\mathcal{C}(\text{lhs}) = \mathcal{C}(\text{rhs})$ □

Theorem 8 (distribution (quantified expression)). *Transformation*

$$\begin{array}{l} \textbf{some } \$x \textbf{ in } (q,\ q') \\ \textbf{satisfies } f(\$x) \end{array} = \begin{array}{l} (\textbf{some } \$x \textbf{ in } q \textbf{ satisfies } f(\$x)) \\ \textbf{or } (\textbf{some } \$x \textbf{ in } q' \textbf{ satisfies } f(\$x)) \end{array} \qquad \text{(TSD)}$$

maintains (*under* (CNSOME)) *or reduces* (*under* (CSOME)) *cost.*

proof for evaluation strategy by (CNSOME) Transformation (TSD) doesn't change cost.

Proof.

$$
\begin{array}{ll}
& \mathcal{C}(\text{lhs}) \\
= & \quad (\text{CNSOME}) \\
& \mathcal{C}(q,\ q') + \mathcal{S}(q,\ q') \cdot \mathcal{C}(f) \\
= & \quad (\text{CSEQ}),(\text{SSEQ}) \\
& \mathcal{C}(q) + \mathcal{C}(q') + (\mathcal{S}(q) + \mathcal{S}(q')) \cdot \mathcal{C}(f) \\
= & \\
& \mathcal{C}(q) + \mathcal{C}(q') + \mathcal{S}(q) \cdot \mathcal{C}(f) + \mathcal{S}(q') \cdot \mathcal{C}(f)
\end{array}
$$

$$
\begin{array}{ll}
& \mathcal{C}(\text{rhs}) \\
= & \quad (\text{COR}) \\
& \mathcal{C}(\textbf{some } \$x \textbf{ in } q \textbf{ satisfies } f(\$x)) + \mathcal{C}(\textbf{some } \$x \textbf{ in } q' \textbf{ satisfies } f(\$x)) \\
= & \quad (\text{CNSOME}) \\
& \mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(f) + \mathcal{C}(q') + \mathcal{S}(q') \cdot \mathcal{C}(f)
\end{array}
$$

Therefore,the cost on both sides is equal. □

Similar arguments apply for universal quantification.

$$
\begin{array}{l}
\textbf{every } \$x \textbf{ in } (q,\ q') \\
\textbf{satisfies } f(\$x)
\end{array}
\quad = \quad
\begin{array}{l}
(\textbf{every } \$x \textbf{ in } q \textbf{ satisfies } f(\$x)) \\
\textbf{and } (\textbf{every } \$x \textbf{ in } q' \textbf{ satisfies } f(\$x))
\end{array}
\qquad (\text{TED})
$$

Proof using early escaping (CSOME) **evaluation strategy** Transformation (TSD) reduces cost.

Proof.

$$
\begin{array}{ll}
& \mathcal{C}(\text{lhs}) \\
= & \quad (\text{CSOME}) \\
& \mathcal{C}((q,q')) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}((q,q'))}}{\mathcal{P}(f)} \cdot \mathcal{C}(f) \\
= & \quad (\text{CSEQ}),(\text{SSEQ}) \\
& \mathcal{C}(q) + \mathcal{C}(q') + \frac{1-(1-\mathcal{P}(f))^{(\mathcal{S}(q)+\mathcal{S}(q'))}}{\mathcal{P}(f)} \cdot \mathcal{C}(f)
\end{array}
$$

$$
\begin{array}{ll}
& \mathcal{C}(\text{rhs}) \\
= & \quad (\text{COR}),(\text{CSOME}) \\
& \mathcal{C}(q) + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(q)}}{\mathcal{P}(f)} \cdot \mathcal{C}(f) \\
& +(1 - \mathcal{P}(\textbf{some } \$y \textbf{ in } q \textbf{ satisfies } f(\$x))) \cdot (\mathcal{C}(q') + \frac{1-(1-\mathcal{P}(f))^{\mathcal{S}(q')}}{\mathcal{P}(f)} \cdot \mathcal{C}(f)) \\
= & \quad (\text{PSOME}) \\
& \mathcal{C}(q) + (1 - \mathcal{P}(f))^{\mathcal{S}(q)} \cdot \mathcal{C}(q') + \frac{1-(1-\mathcal{P}(f))^{(\mathcal{S}(q)+\mathcal{S}(q'))}}{\mathcal{P}(f)} \cdot \mathcal{C}(f)
\end{array}
$$

The cost difference comes from the coefficients of $\mathcal{C}(q')$. As $0 \leq (1 - \mathcal{P}(f)) \leq 1$, it is always smaller than or equal to 1 on the rhs. This implies that the transformation reduces the probability of q' being evaluated, thus leading to a cost reduction. This effect becomes prominent as $\mathcal{P}(f)$ or $\mathcal{S}(q)$ increases. □

Theorem 9 (folding of child axis). *Transformation*

$$\langle QName\rangle\{e\}\langle/QName\rangle/\textbf{child}::* = e \qquad \text{(TC)}$$

reduces cost.

Note that semantic equivalence comes from the fact that child axis access on the lhs yields the content of the direct element constructor. Even though the step expression entails the removal of duplicate nodes and sorting by document order, copy semantics of the element constructor ensures that no duplicate or order skew occurs inside e, thus rhs is simply represented by e.

Proof.

$$\begin{aligned}
&\mathcal{C}(\text{lhs})\\
=\ &\quad \text{(CCSTEP)}\\
&\mathcal{C}(\langle QName\rangle\{e\}\langle/QName\rangle) + k \times \mathcal{S}(\langle QName\rangle\{e\}\langle/QName\rangle)^c\\
=\ &\quad \text{(CDEC),(SDEC)}\\
&\mathcal{C}(e) + \mathcal{S}(e//*) \times C_e + k \times 1^c\\
=\ &\\
&\mathcal{C}(e) + \mathcal{S}(e//*) \times C_e + k
\end{aligned}$$

$\mathcal{C}(\text{rhs}) = \mathcal{C}(e)$

Hence, $\mathcal{C}(\text{lhs}) > \mathcal{C}(\text{rhs})$ □

Other filtering and projection transformation Transformations, such as

$$\begin{aligned}
&(\langle QName_1\rangle\{e_1\}\langle QName_1\rangle, \langle QName_2\rangle\{e_2\}\langle QName_2\rangle)[\textbf{self}::QName_1]\\
=\ &\langle QName_1\rangle\{e_1\}\langle QName_1\rangle
\end{aligned}$$

involve static evaluation of filtering. These transformations obviously simplify expressions, thus always leading to cost reductions. □

β reduction(expansion of let binding)

$$\textbf{let}\ \$x := e_1\ \textbf{return}\ e_2 = e_2[e_1/\$x] \qquad \text{(TLET)}$$

Rhs denotes e_2, with all the occurrences of $\$x$ replaced by e_1.

The evaluation of e_1 takes place as many times as the number of occurrences of $\$x$ inside e_2. So, more than one occurrence of $\$x$ will increase costs. This is exactly the opposite transformation of the common subexpression elimination optimization.

Moreover, if an element construction expression is included within e_1, construction processing itself is copied as well, which leads to different semantics.

Only one occurrence of $\$x$ can avoid an increase in costs. □

Theorem 10 (merger of selection(filter/1)[9]). *Transformation*

$$\begin{array}{lll} \textbf{for } \$x \textbf{ in for } \$y \textbf{ in } q & & \\ \quad \textbf{where } g(\$y) & & \textbf{for } \$x \textbf{ in } q \\ \quad \textbf{return } \$y & = & \textbf{where } g(\$x) \textbf{ and } f(\$x) \\ \textbf{where } f(\$x) & & \textbf{return } \$x \\ \textbf{return } \$x & & \end{array} \qquad \text{(TF1)}$$

doesn't increase cost.

Proof.

$$\begin{array}{ll} & \mathcal{C}(\text{lhs}) \\ = & \quad (\text{CWFOR}),(\text{CVR}) \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot (C_{vr} + \mathcal{C}(g) + \mathcal{P}(g) \cdot \mathcal{C}(\$y) \\ & + \mathcal{S}(\textbf{for } \$y \textbf{ in } q \textbf{ where } g(\$y) \textbf{ return } \$y) \cdot (\mathcal{C}(f) + \mathcal{P}(f) \cdot \mathcal{C}(\$x)) \\ = & \quad (\text{SWFOR}),(\text{CVR}) \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot (C_{vr} + \mathcal{C}(g)) + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot C_{vr} \\ & + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{S}(\$y) \cdot (\mathcal{C}(f) + \mathcal{P}(f) \cdot C_{vr}) \\ = & \quad (\text{SVR}) \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot C_{vr} + \mathcal{S}(q) \cdot \mathcal{C}(g) + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot C_{vr} + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{C}(f) \\ & + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{P}(f) \cdot C_{vr} \end{array}$$

$$\begin{array}{ll} & \mathcal{C}(\text{rhs}) \\ = & \quad (\text{CWFOR}) \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(g(\$x) \textbf{ and } f(\$x)) + \mathcal{P}(g(\$x) \textbf{ and } f(\$x)) \cdot \mathcal{C}(\$x)) \\ = & \quad (\text{CAND}),(\text{PAND}),(\text{CVR}) \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot ((C_{vr} + \mathcal{C}(g) + \mathcal{P}(g) \cdot (C_{vr} + \mathcal{C}(f))) + \mathcal{P}(g) \cdot \mathcal{P}(f) \cdot C_{vr}) \\ = & \\ & \mathcal{C}(q) + \mathcal{S}(q) \cdot C_{vr} + \mathcal{S}(q) \cdot \mathcal{C}(g) + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot C_{vr} + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{C}(f) \\ & + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{P}(f) \cdot C_{vr} \end{array}$$

Thus, costs on both sides are equal. □

Note that the cost of the variable reference is included here, as a return expression consists of only a variable reference. However, ignoring this cost does not affect the equality of the cost estimation. Therefore, the cost of the variable reference is omitted henceforth.

[9] The numbering scheme comes from [15], as it partially inspired our work. We have exploited similar equivalences in XQuery, although not all of them can be mapped to XQuery due to differences in the data model. For example, setmap/4 equivalent XQuery transformation does not exist, because the sequence in XQuery is build on a non-commutative sequence concatenation operator, unlike the union operator for sets.

Theorem 11 (moving selection (filter/2)). *Transformation*

$$\begin{array}{l}\textbf{for } \$x \textbf{ in for } \$y \textbf{ in } q \\ \qquad\qquad \textbf{where } g(\$y) \\ \qquad\qquad \textbf{return } \$y \\ \textbf{return } f(\$x)\end{array} \quad = \quad \begin{array}{l}\textbf{for } \$x \textbf{ in } q \\ \textbf{where } g(\$x) \\ \textbf{return } f(\$x)\end{array} \qquad \text{(TF2)}$$

doesn't change cost.

Proof.

$\mathcal{C}$(lhs)
$=$ (CWFOR),(CFOR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(g) + \mathcal{P}(g) \cdot \mathcal{C}(\$y))$
$+\mathcal{S}(\textbf{for } \$y \textbf{ in } q \textbf{ where } g(\$y) \textbf{ return } \$y) \cdot \mathcal{C}(f)$
$=$ (SWFOR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(g) + \mathcal{P}(g)) + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{S}(\$y) \cdot \mathcal{C}(f)$
$=$ (SVR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(g) + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{C}(f)$

$\mathcal{C}$(rhs)
$=$ (CWFOR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(g) + \mathcal{P}(g) \cdot \mathcal{C}(f))$
$=$
$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(g) + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{C}(f)$
Therefore, the cost is unchanged by the transformation. □

Note that the explicit introduction of the variable reference cost leads to the conclusion of a cost reduction, not a cost increase.

Theorem 12 (filter/3). *Transformation*

$$\begin{array}{l}\textbf{for } \$x \textbf{ in for } \$y \textbf{ in } q \\ \qquad\qquad \textbf{return } f(\$y) \\ \textbf{where } g(\$x) \\ \textbf{return } \$x\end{array} \quad = \quad \begin{array}{l}\textbf{for } \$y \textbf{ in } q \\ \textbf{return} \\ \qquad \textbf{for } \$x \textbf{ in } f(\$y) \\ \qquad \textbf{where } g(\$x) \\ \qquad \textbf{return } \$x\end{array} \qquad \text{(TF3)}$$

doesn't change cost.

Proof.

$\mathcal{C}$(lhs)
$=$ (CWFOR),(CFOR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(f) + \mathcal{S}(\textbf{for } \$y \textbf{ in } q \textbf{ return } f(\$y)) \cdot \mathcal{C}(g)$
$=$ (SWFOR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(f) + \mathcal{S}(q) \cdot \mathcal{S}(f) \cdot \mathcal{C}(g)$

$\mathcal{C}$(rhs)
= (CFOR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(\textbf{for } \$y \textbf{ in } f(\$x) \textbf{ where } g(\$y) \textbf{ return } \$y)$
= (CWFOR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(f(\$x)) + \mathcal{S}(f(\$x)) \cdot (\mathcal{C}(g(\$y)) + \mathcal{P}(g) \cdot \mathcal{C}(\$y)))$
=
$\mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(\$x) + \mathcal{C}(f)) + \mathcal{S}(f) \cdot (\mathcal{C}(\$y) + \mathcal{C}(g))$
=
$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(f) + \mathcal{S}(q) \cdot \mathcal{S}(f) \cdot \mathcal{C}(g)$
Thus, $\mathcal{C}$(lhs) = $\mathcal{C}$(rhs) □

The introduction of an explicit variable reference cost did not change the above equivalence.

Theorem 13 (interchanging selection(filter/5)). *Cost change in transformation*

for $\$x$ **in for** $\$y$ **in** q		**for** $\$x$ **in for** $\$y$ **in** q		
where $g(\$y)$		**where** $f(\$y)$		
return $\$y$	=	**return** $\$y$		(TF5)
where $f(\$x)$		**where** $g(\$x)$		
return $\$x$		**return** $\$x$		

can be expressed using cost of predicates as
$\mathcal{C}(lhs) - \mathcal{C}(rhs) = \mathcal{S}(q) \cdot ((1 - \mathcal{P}(f)) \cdot \mathcal{C}(g) - (1 - \mathcal{P}(g)) \cdot \mathcal{C}(f))$.

Proof.
$\mathcal{C}$(lhs)
= (CWFOR),(SWFOR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot (\mathcal{C}(g(\$y)) + \mathcal{P}(g) \cdot \mathcal{C}(\$y))$
$+(\mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{S}(\$y)) \cdot (\mathcal{C}(f(\$x)) + \mathcal{P}(f) \cdot \mathcal{C}(\$x))$
= (SVR)
$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{C}(g) + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{C}(f)$
=
$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{P}(g) \cdot \mathcal{C}(f) + \mathcal{S}(q) \cdot \mathcal{C}(g)$

$\mathcal{C}$(rhs)
= swap f and g in the lhs
$\mathcal{C}(q) + \mathcal{S}(q) \cdot \mathcal{P}(f) \cdot \mathcal{C}(g) + \mathcal{S}(q) \cdot \mathcal{C}(f)$

$\mathcal{C}$(lhs) − $\mathcal{C}$(rhs) = $\mathcal{S}(q) \cdot ((1 - \mathcal{P}(f)) \cdot \mathcal{C}(g) - (1 - \mathcal{P}(g)) \cdot \mathcal{C}(f))$

Note that the explicit introduction of the variable reference cost didn't change the cost difference.

Therefore, for smaller $\mathcal{P}(f)$ and larger $\mathcal{C}(g)$, or larger $\mathcal{P}(g)$ and smaller $\mathcal{C}(f)$, the cost is reduced by the transformation. □

Corollary 3 *When $\mathcal{C}(g) = \mathcal{C}(f)$, if $\mathcal{P}(g) > \mathcal{P}(f)$ then the cost is reduced by the transformation. That is, placing a high-selectivity filter upstream is cost-beneficial.*

Proof.

$$\begin{aligned}
&\mathcal{C}(\text{lhs}) - \mathcal{C}(\text{rhs}) \\
&= \mathcal{S}(q) \cdot \{(1 - \mathcal{P}(f) - 1 + \mathcal{P}(g))\} \cdot \mathcal{C}(f) \\
&= \mathcal{S}(q) \cdot (\mathcal{P}(g) - \mathcal{P}(f)) \cdot \mathcal{C}(f) \\
&> 0
\end{aligned}$$

□

Corollary 4 *When $\mathcal{P}(g) = \mathcal{P}(f)$, if $\mathcal{C}(g) > \mathcal{C}(f)$, then the transformation reduces cost. That is, placing cheaper filter upstream is cost-beneficial.*

Proof.

$$\mathcal{C}(\text{lhs}) - \mathcal{C}(\text{rhs}) = \mathcal{S}(q) \cdot (1 - \mathcal{P}(f))(\mathcal{C}(g) - \mathcal{C}(f))$$

□

Theorem 14 (if/1). *Transformation*

$$\begin{array}{lcl}
\textbf{if } (e_1) & & \\
\textbf{then if } (e_2) & & \textbf{if } (e_1 \textbf{ and } e_2) \\
\quad \textbf{then } e_3 & = & \textbf{then } e_3 \\
\quad \textbf{else } e_4 & & \textbf{else } e_4 \\
\textbf{else } e_4 & &
\end{array} \qquad \text{(TIF1)}$$

doesn't change cost.

Proof.

$$\begin{aligned}
&\mathcal{C}(\text{lhs}) \\
=\ &\quad \text{(CIF)} \\
&\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(\textbf{if } (e_2) \textbf{ then } e_3 \textbf{ else } e_4) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(e_4) \\
=\ &\quad \text{(CIF)} \\
&\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot (\mathcal{C}(e_2) + \mathcal{P}(e_2) \cdot \mathcal{C}(e_3) \\
&+(1 - \mathcal{P}(e_2)) \cdot \mathcal{C}(e_4)) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(e_4) \\
=\ & \\
&\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(e_2) + \mathcal{P}(e_1) \cdot \mathcal{P}(e_2) \cdot \mathcal{C}(e_3) \\
&+\mathcal{P}(e_1) \cdot (1 - \mathcal{P}(e_2)) \cdot \mathcal{C}(e_4) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(e_4) \\
=\ & \\
&\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(e_2) + \mathcal{P}(e_1) \cdot \mathcal{P}(e_2) \cdot \mathcal{C}(e_3) + (1 - \mathcal{P}(e_1) \cdot \mathcal{P}(e_2)) \cdot \mathcal{C}(e_4)
\end{aligned}$$

$$\begin{aligned}
&\mathcal{C}(\text{rhs}) \\
=\ &\quad \text{(CIF)} \\
&\mathcal{C}(e_1 \textbf{ and } e_2) + \mathcal{P}(e_1 \textbf{ and } e_2) \cdot \mathcal{C}(e_3) + (1 - \mathcal{P}(e_1 \textbf{ and } e_2)) \cdot \mathcal{C}(e_4) \\
=\ &\quad \text{(CAND),(PAND)} \\
&\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(e_2) + \mathcal{P}(e_1) \cdot \mathcal{P}(e_2) \cdot \mathcal{C}(e_3) + (1 - \mathcal{P}(e_1) \cdot \mathcal{P}(e_2)) \cdot \mathcal{C}(e_4)
\end{aligned}$$

Therefore, $\mathcal{C}(\text{lhs}) = \mathcal{C}(\text{rhs})$ □

Theorem 15 (if/4). *Transformation*

$$f(\textbf{if } (e_1) \textbf{ then } e_2 \textbf{ else } e_3) \;=\; \textbf{if } (e_1) \textbf{ then } f(e_2) \textbf{ else } f(e_3) \qquad \text{(TIF4)}$$

doesn't change cost.

Proof.

$\mathcal{C}(\text{lhs})$
$=$ (CFCALL)
$\mathcal{C}(\textbf{if } (e_1) \textbf{ then } e_2 \textbf{ else } e_3) + \mathcal{C}(f)$
$=$ (CIF)
$\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(e_2) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(e_3) + \mathcal{C}(f)$

$\mathcal{C}(\text{rhs})$
$=$ (CIF)
$\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(f(e_2)) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(f(e_3))$
$=$ (CFCALL)
$\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot (\mathcal{C}(e_2) + \mathcal{C}(f)) + (1 - \mathcal{P}(e_1)) \cdot (\mathcal{C}(e_3) + \mathcal{C}(f))$
$=$
$\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(e_2) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(e_3) + \mathcal{C}(f)$
Therefore, $\mathcal{C}(\text{lhs}) = \mathcal{C}(\text{rhs})$ □

4.3 Summary

In this section, the cost changes in various transformation rules are proved based on the proposed cost model. These proofs guarantee that by constructing optimization, using only the cost-non-increasing subsets of these transformations, keep or decrease costs, because each transformation applied in this optimization will keep or improve costs.

5 Related works on cost models

In this section, we describe the preceding work related to the XQuery cost model.

Cost models must be based on a physical plan. We have already discussed this subject in Section 2. For implementations in which XML data is stored in RDB and XQuery is transformed to SQL, the cost models are discussed based on those of SQL engines.

In [22], published at the initial stage of development of the formal semantics, the cost models were supposed to be developed for each physical storage. Optimizing XQuery while keeping the semantics of the query unchanged, which was considered a difficult task, was expected to be supported by formal semantics. We have actually quoted formal semantics within the context of optimization. However, the cost model was not determined in that paper, probably because Galax was not tightly coupled with physical storage at that time.

In [23], which describes the XQuery processor in streaming context, a cost model is explicitly avoided. Reasons are also mentioned, which are quoted here along with our opinion (following →).

– From a stream processing standpoint, cost model development in the world of the Internet is difficult, because statistical information is unpredictable. → As we are using a relative cost model (do not calculate concrete values), we are in a position where discussions that are independent of the data source are possible.
– Development of an effective cost model itself is difficult for an XQuery engine. → We are not dealing with run-time plan selection based on a cost model, which is inherently difficult. Instead, we calculate the gain or loss that can be statically determined.
– Role of the cost model for XQuery is smaller than that of SQL. As the XQuery data model prohibits reordering (which seems to refer to noncommutativity of the sequence constructor ","), there is much less freedom in transformations, so it is less effective. Good heuristics are more important for covering the lack of a cost model.
  → As we have enumerated in this paper, there are many transformation rules in spite of the lack of freedom of permutations. We had to determine for these transformations the relative loss or gain that is independent of the input data, so the cost model is quite important.
– The cost model turned out to be unimportant for the real-world applications that the authors had dealt with.
  → We have encountered the behavior of engines (a change of execution speed that depends on the syntactic position of a selection), that made us recognize the importance of the cost model. Although each of these may seem heuristics, the cost calculation of the arbitrarily nested XQuery expressions may require a decision that is beyond heuristics.

[24] deals with the optimization based on a cost model within the context of XML data stored in RDB. The cost model is based on the selectivity of predicates as well as statistical information. Specifically, the cost model for RDB [25] is adopted. Conversely, [25] mentions as their target of optimization the relational query which is transformed from XML-QL. Elaborated cost formulas are described.

Although a concrete cost model is not mentioned in [26], it provides a bird's-eye view of the cost model, in the context of transaction processing, by classifying the level of cost discussion in language, logical access, and storage. RDB seems to be assumed as a back-end.

TIMBER [27] translates queries including XQuery into their tree algebra, in which cost estimator plays an important role. An algorithm to generate query execution plans to derive optimal pipeline processing is proposed. The grounding cost function uses the estimation of selectivity in a path expression [3] and the size function [4]. [4] does not handle XQuery as a whole, but deals with the estimates for twig queries. That is, they estimate portions that match the tree patterns. Their focus includes concrete operations like join.

We are dealing with XQuery in general, including FLWOR and the quantified expression. Although there are prior research conducted on selectivity estimation and size function in general, we can safely leave the subcomponent of the body of

the cost functions uninterpreted, as long as they do not participate in the transformation, i.e., they remain unchanged throughout the transformation. Thus we can discuss the costs independently of the return value of those functions.

6 Discussions

6.1 Cost model with respect to query size

We have not in general specified the cost with respect to the query size, because unlike XPath, a simple measure of the query size is not available.

However, a cost model combined with a query size can be derived from our proposed cost functions from various points of view.

For example, if the query size is defined in terms of the nesting depth of **for** expressions, the combined cost function can be obtained by applying the original cost function on nested **for** expressions with depth n. Under the definition of the query size in terms of the number of cascading **for** expressions, the combined cost functions can be similarly obtained.

6.2 Estimating gain for whole expression

As a concrete example, we have calculated the cost difference of a sample query that consists of XMark[10] [28] Q10 and the query on it. Details are described in Appendix B.

7 Conclusion

In this paper, within the context of XQuery optimization based on source level query rewriting, a cost model is defined to quantify the costs of expressions, upon which a relative cost change for every transformation rule described in this paper is provided.

Cost models have to reflect engines' evaluation strategies. During our cost model construction, formal semantics were adopted for expressions that are expected to have evaluation strategies in formal semantics. For expressions whose efficient evaluation strategies were proposed in the literature, the specific strategies were adopted. For the cost of element construction that is not apparent in the formal semantics, the cost modeling was conducted quoting experiments on real engines. It has been proved that as long as rewriting consists of transformation rules that do not degrade the cost, rewriting on the whole can be guaranteed to improve costs. Since real world engines, as well as specifications themselves are still unstable, we expect our cost model to play the role of "virtual engine" as a simple evaluation tool for XQuery optimization research.

Acknowledgements We are grateful to Dr. Yasunori Ishihara at Osaka University for his valuable comments and suggestions.

[10] `http://www.ins.cwi.nl/projects/xmark/Assets/xmlquery.txt`

References

1. Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML Query Language. W3C Working Dract (2004) http://www.w3.org/TR/2004/WD-xquery-20040723/
2. Koch, C.: On the Complexity of Nonrecursive XQuery and Functional Query Languages on Complex Values. In: PODS. (2005)
3. Aboulnaga, A., Alameldeen, A.R., Naughton, J.F.: Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In: VLDB. (2001) 591–600
4. Wu, Y., Patel, J.M., Jagadish, H.V.: Estimating Answer Sizes for XML Queries. In: EDBT. (2002) 590–608
5. Grinev, M., Kuznetsov, S.: Towards an Exhaustive Set of Rewriting Rules for XQuery Optimization: BizQuery Experience. In Manolopoulos, Y., Nàvrat, P., eds.: ADBIS 2002. Volume 2435 of LNCS., Springer-Verlag (2002) 340–345
6. Fernandez, M.F., Siméon, J., Wadler, P.: A semi-monad for semi-structured data. In: Proceedings of the 8th International Conference on Database Theory, Springer-Verlag (2001) 263–300
7. Manolescu, I., Florescu, D., Kossmann, D.: Pushing XML Queries inside Relational Databases. Technical Report RR-4112, INRIA (2001)
8. Deutsch, A., Papakonstantinou, Y., Xu, Y.: The NEXT Logical Framework for XQuery. In: VLDB. (2004) 168–179
9. Fernandez, M.F., Tan, W.C., Suciu, D.: SilkRoute: trading between relations and XML. Computer Networks **33** (2000) 723–745
10. Draper, D., Fankhauser, P., Fernández, M., Malhotra, A., Rose, K., Rys, M., Siméon, J., Wadler, P.: XQuery 1.0 and XPath 2.0 Formal Semantics. W3C Working Draft (2004) http://www.w3.org/TR/2004/WD-xquery-semantics-20040220/
11. Fegaras, L., Maier, D.: Optimizing object queries using an effective calculus. ACM Trans. Database Syst. **25** (2000) 457–516
12. Manolescu, I., Florescu, D., Kossmann, D.: Answering XML Queries on Heterogeneous Data Sources. In: VLDB. (2001) 241–250
13. Hidders, J., Paredaens, J., Vercammen, R., Demeyer, S.: A Light but Formal Introduction to XQuery. In: Proceedings of the Second International XML Database Symposium (XSym04). LNCS, Toronto, Canada, Springer (2004) 5–20
14. Kato, H., Hidaka, S., Yoshikawa, M.: FLWOR Arranging; XQuery Partial Evaluation by Child-path Folding(to be submitted). Technical report, National Institute of Informatics (2005)
15. Poulovassilis, A., Small, C.: Algebraic Query Optimisation for Database Programming Languages. VLDB Journal **5** (1996) 119–132
16. Brundage, M.: XQuery: The XML Query Language. Pearson Higher Education (2004)
17. Katz, H., Chamberlin, D., Draper, D., Fernández, M., Kay, M., Robie, J., Rys, M., Siméon, J., Tivy, J., Wadler, P.: XQuery from the Experts: A Guide to the W3C XML Query Language. Addison-Wesley Longman Publishing Co., Inc. (2003)
18. Fahringer, T.: The Weight Finder – An Advanced Profiler for Fortran Programs. In Keßler, C.W., ed.: Automatic Parallelization. Vieveg Advanced Studies in Computer Science (1994) 7–31
19. Lucent: Galax (2003) http://db.bell-labs.com/galax/
20. Meier, W.: eXist: An Open Source Native XML Database. In: Proceedings of the Web- and Database-Related Workshops. LNCS, Erfurt, Germany, Springer (2002) 169–183

21. Gottlob, G., Koch, C., Pichler, R.: Efficient Algorithms for Processing XPath Queries. In: VLDB. (2002) 95–106
22. Choi, B., Fernández, M., Siméon, J.: The XQuery Formal Semantics: A Foundation for Implementation and Optimization. Technical Report MS-CIS-02-25, University of Pennsylvania (2002)
23. Florescu, D., Hillery, C., Kossmann, D., Lucas, P., Riccardi, F., Westmann, T., Carey, M.J., Sundararajan, A.: The BEA Streaming XQuery Processor. VLDB Journal **13** (2004) 294–315
24. Bohannon, P., Freire, J., Roy, P., Siméon, J.: From XML Schema to Relations: A Cost-Based Approach to XML Storage. In: ICDE. (2002) 64–75
25. Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S.: Efficient and extensible algorithms for multi query optimization. In: SIGMOD Conference. (2000) 249–260
26. Mathis, C., Häder, T.: A Query Processing Approach for XML Database Systems. In: Workshop über Grundlagen von Datenbanke. (2005)
27. Jagadish, H.V., Al-Khalifa, S., Chapman, A., Lakshmanan, L.V.S., Nierman, A., Paparizos, S., Patel, J.M., Srivastava, D., Wiwatwattana, N., Wu, Y., Yu, C.: TIMBER: A native XML database. VLDB Journal **11** (2002) 274–291
28. Schmidt, A., Waas, F., Kersten, M.L., Carey, M.J., Manolescu, I., Busse, R.: XMark: A benchmark for XML data management. In: VLDB. (2002) 974–985

A Proofs of semantic equivalences of transformations

A.1 Preliminaries

Transformations and free variables For all expressions that include free variables, corresponding environment which uniquely determines the bindings of the variables are assumed.

Statement $Expr_1 = Expr_2$ means that if both sides are evaluated under identical environments, equivalence is guaranteed in any of such environments. That is, let the variable binding environment be Γ,

$$\frac{\forall \Gamma \left(\begin{array}{c} \Gamma \vdash Expr_1 \text{ yields } Value_1 \\ \Gamma \vdash Expr_2 \text{ yields } Value_2 \\ Value_1 == Value_2 \end{array} \right)}{\forall \Gamma \vdash Expr_1 = Expr_2}$$

Proof of this statement consists of the reduction of expressions on both sides into an identical expression.

For variables appearing on both sides, that have identical names, under the above assumption that both are provided with identical environments, identical values are bound. Therefore, identical expression is evaluated to identical values, and consequently, the existence of free variables does not hinder the validity of the statement.

Φ_Q introduction to unify some and for expressions To uniformly treat **for** and quantified expressions, a binary reduction operator $\Phi_Q$is introduced as follows.

$\Phi_Q :: (a \to b) \to (b \to b \to b) \to b \to (a) \to b$

definition

(1) $\Phi_Q\ f\ op\ e\ ()\ = e$
(2) $\Phi_Q\ f\ op\ e\ (x)\ = f\ x$
(3) $\Phi_Q\ f\ op\ e\ (q\ ,\ q')\ = (\Phi_Q\ f\ op\ e\ q)$
$op\ (\Phi_Q\ f\ op\ e\ q')$

For the type of q, a sequence of arbitrary type a is assumed. Associativity of the operator op is also assumed. In the above definition, (1) is for an empty input, (2) is for a singleton input, and (3) is for a concatenation of the two input sequences.

Typing rule

$$\frac{f :: a \to b \quad op :: b \to b \to b \quad e :: b \quad q :: (a)}{(\Phi_Q\ f\ op\ e\ q) :: b}\ ,$$

where $e :: T$ denotes that expression e has type T.

Theorem 16 ($\Phi_Q \leftrightarrow$ for expression correspondence).

$$(\textbf{for}\ \$x\ \textbf{in}\ q\ \textbf{return}\ f(\$x)) = (\Phi_Q\ f(\$x)\ ,\ ()\ q)$$

Proof. **(1)** for empty input:
It is apparent from formal semantics that

$$\textbf{for } \$x \textbf{ in } () \textbf{ return } f(\$x) \;\Rightarrow\; () = (\Phi_Q\ f\ op\ ,\ ()\ ())$$

(2) for singleton input:
According to formal semantics with input length equal to 1

$$\textbf{for } \$x \textbf{ in } y \textbf{ return } f(\$x) \;\Rightarrow\; f(\$y) = (\Phi_Q\ f\ op\ ()\ (y))$$

(3) for arbitrary sequence q

Proof by induction.
Induction hypothesis:

$$\textbf{for } \$x \textbf{ in } q \textbf{ return } f(\$x) = \Phi_Q\ f\ ,\ ()\ q$$

for (q, y) (y denotes a singleton)

$$\begin{array}{ll} & \textbf{for } \$x \textbf{ in } (q, y) \textbf{ return } f(\$x) \\ = & \quad \text{formal semantics of } \textbf{for} \\ & f(q_1), f(q_2), \ldots, f(q_N), f(y) \\ = & \quad (2) \text{ above} \\ & (\textbf{for } \$x \textbf{ in } q \textbf{ return } f(\$x)), (\Phi_Q\ f\ ,\ ()\ y) \\ = & \quad \text{induction hypothesis} \\ & ((\Phi_Q\ f\ ,\ ()\ q),\ (\Phi_Q\ f\ ,\ ()\ y)) \\ = & \quad \text{definition (3) of } \Phi_Q \\ & (\Phi_Q\ f\ ,\ ()\ (q,\ (y))) \end{array}$$

i.e., if q' satisfies

$$\textbf{for } \$x \textbf{ in } q' \textbf{ return } f(\$x) = (\Phi_Q\ f\ ,\ ()\ q')$$

then (q', y) satisfies

$$\textbf{for } \$x \textbf{ in } (q',\ y) \textbf{ return } f(\$x) = ((\Phi_Q\ f\ ,\ ()\ (q',\ (y))))$$

By the way, for empty and singleton input sequence q,

$$\textbf{for } \$x \textbf{ in } q \textbf{ return } f(\$x) = \Phi_Q\ f(\$x)\ ,\ ()\ q$$

holds. Therefore, it is inductively proved that

$$\textbf{for } \$x \textbf{ in } q \textbf{ return } f(\$x) = \Phi_Q\ f(\$x)\ ,\ ()\ q.$$

□

Theorem 17 ($\Phi_Q \leftrightarrow$ some/every correspondence).

$$\textbf{some } \$x \textbf{ in } q \textbf{ satisfies } f(\$x)\ = (\Phi_Q\ f\ \vee\ \mathsf{false}\ q)$$

Proof. **(1) empty input**

$$\textbf{some } \$x \textbf{ in } () \textbf{ satisfies } f(\$x) \;= \mathsf{false} \;= (\Phi_Q\ f\ \vee\ \mathsf{false}\ ())$$

Formal semantics with negation of the condition of at least one of the input makes $f()$ yield true.

(2) singleton input

$$\textbf{some } \$x \textbf{ in } y \textbf{ satisfies } f(\$x) \;= f(y) = (\Phi_Q\ f\ \vee\ \mathsf{false}\ (y))$$

Formal semantics with only input y that solely depends on its application to $f(y)$.

(3) for arbitrary sequence input q

Proof by induction.

Induction hypothesis:

$$\textbf{some } \$x \textbf{ in } q \textbf{ satisfies } f(\$x) \;= (\Phi_Q\ f\ \vee\ \mathsf{false}\ q)$$

For input sequence q with singleton y concatenated

$$\begin{aligned}
&\textbf{some } \$x \textbf{ in } (q, y) \textbf{ satisfies } f(\$x)\\
=\ &\text{formal semantics}\\
&f(q_1)\ \vee\ f(q_2)\ \vee\ \cdots \vee\ f(q_N)\ \vee\ f(y)\\
=\ &\text{formal semantics, (2) above}\\
&\textbf{some } \$x \textbf{ in } q \textbf{ satisfies } f(\$x)\ \vee\ (\Phi_Q\ f\ \vee\ \mathsf{false}\ (y))\\
=\ &\text{induction hypothesis}\\
&(\Phi_Q\ f\ \vee\ \mathsf{false}\ q)\ \vee\ (\Phi_Q\ f\ \vee\ \mathsf{false}\ (y))\\
=\ &\text{definition (3) of } \Phi_Q\\
&(\Phi_Q\ f\ \vee\ \mathsf{false}\ (q, (y)))
\end{aligned}$$

For **every** expression, similar arguments apply which result in

$$\textbf{every } \$x \textbf{ in } q \textbf{ satisfies } f(\$x) \;= (\Phi_Q\ f\ \wedge\ \mathsf{true}\ q)$$

□

A.2 Proofs of tranformation rules

TR 1 (where $\leftrightarrow$ if)

$$\begin{aligned}
&\textbf{for } \$x \textbf{ in } q \textbf{ where } f(\$x) \textbf{ return } g(\$x)\\
=\ &\textbf{for } \$x \textbf{ in } q \textbf{ return if } f(\$x) \textbf{ then } g(\$x) \textbf{ else } ()
\end{aligned} \qquad \text{(TWIF)}$$

Proof. Normalization rule of For expressions in Formal semantics □

TR 2 (for left unit (TFLU)**)**

Direct consequence of the second definition of Φ_Q and $\Phi_Q\leftrightarrow$ **for** expression correspondence. □

TR 3 (quantified left unit (Tslu),(Telu)**)**

Direct consequence of the 2nd definition of Φ_Q and Theorem 17. □

TR 4 (for right unit (Tfru)**)**

Proof. For $q = (q_1, q_2, \ldots, q_N)$, by Formal Semantics:

$$\textbf{for } \$x \textbf{ in } q \textbf{ return } \$x = q_1, q_2, \ldots, q_N = q$$

□

TR 5 (quantified distributive (Tsd),(Ted)**)**

Direct consequence of the 3rd definition of Φ_Q and Theorem 17. □

Theorem 18 ($\Phi_Q$associative).

$$\Phi_Q\ f\ op\ e\ (\Phi_Q\ g\ ,\ ()\ q) = \Phi_Q\ (\Phi_Q\ f\ op\ e\ g)\ op\ e\ q \qquad \text{(Tpa)}$$

Proof.

$$\begin{array}{ll} & \text{lhs} \\ = & \text{definition of } \Phi_Q \\ & \Phi_Q\ f\ op\ e\ (g(q_1), g(q_2), \ldots, g(q_N)) \\ = & \text{3rd definition of } \Phi_Q \text{ (quantified distributive)} \\ & (\Phi_Q\ f\ op\ e\ g(q_1))\ op\ (\Phi_Q\ f\ op\ e\ g(q_2))\ op \ldots op\ (\Phi_Q\ f\ op\ e\ g(q_N)) \\ = & \text{3rd definition of } \Phi_Q \text{ (rhs to lhs)} \\ & \Phi_Q\ (\Phi_Q\ f\ op\ e\ g)\ op\ e\ q \end{array}$$

Note that even if g is in the scope of the variable binding of outer Φ_Q at the rhs, g in the lhs is not. Consequently, g should not include the bound variable introduced by the outer Φ_Q. □

TR 6 (for associative (Tfa)**)**

Substitute *op* and e in (Tpa) with , and (), respectively.

TR 7 (some/every associative (Tsa),(Tea)**)**

Substitute *op* and e in (Tpa) with $\vee$ and false in case of **some**, $\wedge$ and true in case of **every**, respectively.

TR 8 (if/4 (Tif4)**)**

$$f(\textbf{if } (e_1) \textbf{ then } e_2 \textbf{ else } e_3) = \textbf{if } (e_1) \textbf{ then } f(e_2) \textbf{ else } f(e_3)$$

Proof.

$$
\begin{array}{ll}
 & \text{lhs} \\
= & \\
 & \textbf{if } (e_1) \textbf{ then } f(\textbf{if } (\mathsf{true}) \textbf{ then } e_2 \textbf{ else } e_3) \textbf{ else } f(\textbf{if } (\mathsf{false}) \textbf{ then } e_2 \textbf{ else } e_3) \\
= & \quad \text{definition of } \textbf{if} \\
 & \textbf{if } (e_1) \textbf{ then } f(e_2) \textbf{ else } f(e_3)
\end{array}
$$

□

TR 9 (if/1 (TIF1)**)**

$$\textbf{if } e_1 \ (\textbf{if } e_2 \ e_3 \ e_4) \ e_4) = \textbf{if } (e_1 \textbf{ and } e_2) \textbf{ then } e_3 \textbf{ else } e_4$$

Proof.

$$
\begin{array}{ll}
 & \text{lhs} \\
= & \\
 & \textbf{if } (e_1 \textbf{ and } e_2) \textbf{ then} \\
 & (\textbf{if } (\mathsf{true}) \ (\textbf{if } (\mathsf{true}) \textbf{ then } e_3 \textbf{ else } e_4) \textbf{ else } e_4) \\
 & \textbf{else } (\textbf{if } (e_1) \ (\textbf{if } \mathsf{false} \textbf{ then } e_3 \textbf{ else } e_4) \textbf{ else } e_4) \\
= & \quad \text{definition of } \textbf{if} \\
 & \textbf{if } (e_1 \textbf{ and } e_2) \textbf{ then } e_3 \textbf{ else } (\textbf{if } (e_1) \textbf{ then } e_4 \textbf{ else } e_4) \\
= & \quad \text{property of } \textbf{if} \\
 & \textbf{if } (e_1 \textbf{ and } e_2) \textbf{ then } e_3 \textbf{ else } e_4
\end{array}
$$

□

TR 10 (Left unit law of for expression with where clause)

Proof. for singleton y,

$$
\begin{array}{ll}
 & \textbf{for } \$x \textbf{ in } y \textbf{ where } f(\$x) \textbf{ return } g(\$x) \\
= & \quad (\textsc{Twif}) \\
 & \textbf{for } \$x \textbf{ in } y \textbf{ return if } (f(\$x)) \textbf{ then } g(\$x) \textbf{ else } () \\
= & \quad (\textsc{Tflu}) \\
 & \textbf{if } (f(y)) \textbf{ then } g(y) \textbf{ else } () \qquad\qquad (\textsc{Twflu})
\end{array}
$$

□

TR 11 (associative law of for with where clause (TWFA)**)**

Proof.

lhs
= **for** $\$x$ **in** (**for** $\$y$ **in** q
where $e(\$y)$ **return** $f(\$y)$) **where** $g(\$x)$ **return** $h(\$x)$
= (TWIF)
for $\$x$ **in for** $\$y$ **in** q **return if** $(e(\$y))$ **then** $h(\$y)$ **else** ()
return if $(g(\$x))$ **then** $h(\$x)$ **else** ()
= (TFA)
for $\$y$ **in** q **return for** $\$x$ **in** (**if** $(e(\$y))$ **then** $h(\$y)$ **else** ())
return if $(g(\$x))$ **then** $h(\$x)$ **else** ()
= (TIF4)
for $\$y$ **in** q **return if** $(e(\$y))$ **then**
(**for** $\$x$ **in** $h(\$y)$ **return if** $(g(\$x))$ **then** $h(\$x)$ **else** ()
else (**for** $\$x$ **in** () **return if** $(g(\$x))$ **then** $h(\$x)$ **else** ()
= **for** with empty input
for $\$y$ **in** q **return if** $(e(\$y))$ **then**
(**for** $\$x$ **in** $h(\$y)$ **return if** $(g(\$x))$ **then** $h(\$x)$ **else** ()
else ()
= (TWIF)
for $\$y$ **in** q **where** $(e(\$y))$
return for $\$x$ **in** $h(\$y)$ **where** $(g(\$x))$ **return** $h(\$x)$

□

TR 12 (filter/0) [11]

This transformation pulls filtering $h(\$x)$ up.

for $\$x$ **in for** $\$y$ **in** q **where** $g(\$y)$ **return** $\$y$ **where** $h(\$x)$ **return** $f(\$x)$
= **for** $\$y$ **in** q **where** $(g(\$y)$ **and** $h\$(x))$ **return** $f(\$x)$ (TF0)

Proof.

lhs
= (TWFA)
for $\$y$ **in** q **where** $g(\$y)$ **return for** $\$x$ **in** $\$y$ **where** $h(\$x)$ **return** $f(\$x)$
= (TWFLU)

[11] This transformation is introduced for the purpose of simplification of the proofs.

for $y **in** q **where** $g(\$y)$ **return if** $(h\$(y))$ **then** $h(\$y)$ **else** ()

= (TWIF)

for $y **in** q **return if** $(g(\$y))$ **then** (**if** $(h\$(y))$ **then** $h(\$y)$ **else** ()) **else** ()

= (TIF1)

for $y **in** q **return if** $(g(\$y)$ **and** $h\$(y))$ **then** $h(\$y)$ **else** ()

= (TWIF), reverse

for $y **in** q **where** $(g(\$y)$ **and** $h\$(y))$ **return** $f(\$y)$

□

TR 13 (filter/1(TF1)**)**

Proof.

lhs

=

for $x **in for** $y **in** q **where** $g(\$y)$ **return** $y **where** $f(\$x)$ **return** $x

= (TF0)

for $y **in** q **where** $(g(\$y)$ **and** $f(\$y))$ **return** $y

=

rhs

□

TR 14 (filter/2(TF2)**)**

Proof.

lhs

=

for $x **in for** $y **in** q **where** $g(\$y)$ **return** $y **return** $f(\$x)$

= dummy where introduction

for $x **in for** $y **in** q **where** $g(\$y)$ **return** $y **where** true **return** $f(\$x)$

= (TF0)

for $y **in** q **where** $(g(\$y)$ **and** true**)** **return** $f(\$y)$

= property of **and**

for $y **in** q **where** $g(\$y)$ **return** $f(\$y)$

□

TR 15 (filter/3(TF3)**)**

Proof.

lhs

=

for $x **in for** $y **in** q **return** $f(\$y)$ **where** $g(\$x)$ **return** $x

= (TWFA)

for $y **in** q **return for** $x **in** $f(\$y)$ **where** $g(\$x)$ **return** $x

□

TR 16 (filter/5(TF5)**)**

Proof.

lhs
=
for $x **in** (**for** $y **in** q **where** $g(\$y)$ **return** $\$y$) **where** $f(\$x)$ **return** $\$x$
= (TF1)
for $\$y$ **in** q **where** $(g(\$y)$ **and** $(f(\$y)))$ **return** $\$y$
= property of **and**
for $\$y$ **in** q **where** $(h(\$y)$ **and** $(g(\$y)))$ **return** $\$y$
= (TF1) (rhs→ lhs)
for $\$x$ **in for** $\$y$ **in** q **where** $h(\$y)$ **return** $\$y$ **where** $g(\$x)$ **return** $\$x$
=
rhs

□

B Concrete example of cost calculation

The cost estimation for a concrete sample of XQuery is described in this section. The query consists of XMark Q10 as a view definition, and another query to the view definition that picks up cities where citizens with more than 10000 income resides.

```
(: Q10: Grouping :)
(: Q10: List all persons according to their interest; use French markup in the result. :)
let $view := (
for $i in distinct-values($auction/site/people/person/profile/interest/@category)
let $p := for    $t in $auction/site/people/person
          where  $t/profile/interest/@category = $i
            return
              <personne>
                <statistiques>
                  <sexe> { $t/profile/gender/text() } </sexe>
                  <age> { $t/profile/age/text() } </age>
                  <education> { $t/profile/education/text() } </education>
                  <revenu> { fn:data($t/profile/@income) } </revenu>
                </statistiques>
                <coordonnees>
                  <nom> { $t/name/text() } </nom>
                  <rue> { $t/address/street/text() } </rue>
                  <ville> { $t/address/city/text() } </ville>
                  <pays> { $t/address/country/text() } </pays>
                  <reseau>
                    <courrier> { $t/emailaddress/text() } </courrier>
                    <pagePerso> { $t/homepage/text() } </pagePerso>
                  </reseau>
                </coordonnees>
                <cartePaiement> { $t/creditcard/text() } </cartePaiement>
              </personne>
return <categorie>
         <id> { $i } </id>
         { $p }
       </categorie>
)
for $p in $view/personne
where $p/statistiques/revenu > 10000
return $p/coordonnees/ville
```

After optimization:

```
for $i in distinct-values($auction/site/people/person/profile/interest/@category)
for $t in $auction/site/people/person
where ($t/profile/interest/@category = $i) and
      (<revenu> { fn:data($t/profile/@income) } </revenu> > 10000)
return <ville> { $t/address/city/text() } </ville>
```

Results of the cost reduction estimations of this optimization are as follows:

$+\mathcal{S}$(distinct-values($auction/site/people/person/profile/interest/@category))
 $\cdot$($\mathcal{S}$($auction/site/people/person)$\cdot\mathcal{P}$($t/profile/interest/@category = $i)
  $\cdot$($\mathcal{C}$(<personne>
        ...
        <revenu> {fn:data($t/profile/@income) } </revenu> (: 1 :)
        ...
        <ville> { $t/address/city/text() } </ville> (: 2 :)
        ...
        </personne>)

      $-\mathcal{C}$(<revenu> { fn:data($t/profile/@income) } </revenu>) (: 1' :)

      $-\mathcal{P}$(<revenu> { fn:data($t/profile/@income) } </revenu> > 10000)
      $\cdot\mathcal{C}$(<ville> { $t/address/city/text() } </ville>) (: 2' :)

    $+$($\mathcal{C}$($p1/statistiques/revenu)
      $+\mathcal{P}$($p1/statistiques/revenu > 10000)
      $\cdot\mathcal{C}$($p1/coordonnees/ville))
)
  $+\mathcal{C}$(<categorie><id>{$i}</id>{$p}</categorie>)
)
$+\mathcal{C}$($view/personne)

Note that (: 1 :) and (: 2 :) cancel with (: 1' :) and (: 2' :) (2' has a coefficient which is smaller than one), respectively, leaving only positive values. Consequently, the above transformation is guaranteed to speed up query evaluations. Quantitatively, the result shows that the reduction is proportional to the 2nd power of the document size (s). Actual query evaluation time using real engine is depicted in Fig. 18.

The experiment was conducted on a 1.5GHz quad Xeon SMP machine running Linux kernel 2.4.20. Time reduction (T) can be measured by subtracting opt from normal, which fits to $T = 0.27s^{1.9}$ using a regression analysis[12]. This experimental result shows that the cost model can actually simulate the effect of optimization for a real engine.

[12] The ratio of the difference between the corrected total sum of squares (with intercept term) and the residual sum of squares to the corrected total sum of squares, which is a possible definition of R^2 for a nonlinear regression, was 1.0.

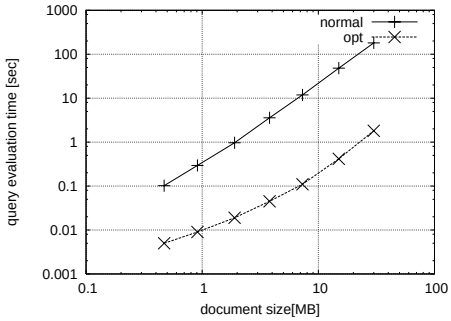

Fig. 18. Query for Q10 of XMark (Galax 0.4.0 main memory)

expression	cost	label	size	label
for $\$x$ **in** s **return** $f(\$x)$	$\mathcal{C}(s) + \mathcal{S}(s) \cdot \mathcal{C}(f)$	CFOR	$\mathcal{S}(s) \cdot \mathcal{S}(f)$	SFOR
for $\$x$ **in** s **where** $p(\$x)$ **return** $f(\$x)$	$\mathcal{C}(s) + \mathcal{S}(s) \cdot (\mathcal{C}(p) + \mathcal{P}(p) \cdot \mathcal{C}(f))$	CWFOR	$\mathcal{S}(s) \cdot \mathcal{P}(p) \cdot \mathcal{S}(f)$	SWFOR
for $\$x_1$ **in** s_1, $\$x_2$ **in** $s_2(\$x_1)$ **return** $f(\$x)$	$\mathcal{C}(s_1) + \mathcal{S}(s_1) \cdot \mathcal{C}(s_2) + \mathcal{S}(s_1) \cdot \mathcal{S}(s_2) \cdot \mathcal{C}(f)$		$\mathcal{S}(s_1) \cdot \mathcal{S}(s_2) \cdot \mathcal{P}(p) \cdot \mathcal{S}(f)$	
literal	constant		(uninterpreted)	
$\langle QName \rangle \{e\} \langle /QName \rangle$	$\mathcal{C}(e) + \mathcal{S}(e//*) \cdot C_e$	CDEC	1	SDEC
element $QName$ $\{e\}$		CCEC		SCEC
$e/QName$	$\mathcal{C}(e) + \mathcal{S}(e)^c \cdot k$	CCSTEP	$k \cdot \mathcal{P}(\text{nodename} == QName) \cdot \mathcal{S}(e)$	SCSTEP
$e/q_1/q_2/\ldots/q_N$	$\mathcal{C}(e) + \mathcal{S}(e)^c \times N^d$	CGSTEP		
$e//QName$	$\mathcal{C}(e) + k \cdot \mathcal{S}(e//*)$	CDSTEP	$\mathcal{S}(e//*) \cdot \mathcal{P}(\text{nodename} == QName)$	SDSTEP
e_1, e_2	$\mathcal{C}(e_1) + \mathcal{C}(e_2)$	CSEQ	$\mathcal{S}(e_1) + \mathcal{S}(e_2)$	SSEQ
let $\$x := s$ **return** $f(\$x)$	$\mathcal{C}(s) + \mathcal{C}(f)$	CLET	$\mathcal{S}(f)$	SLET
$\$x$	constant $= C_{vr}$	CVR	$\mathcal{S}(\text{dereference}(\$x))$	SVR
(e)	$\mathcal{C}(e)$		$\mathcal{S}(e)$	SPAREN
if (e_1) **then** e_2 **else** e_3	$\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(e_2) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(e_3)$	CIF	$\mathcal{P}(e_1) \cdot \mathcal{S}(e_2) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{S}(e_3)$	SIF
e_1 **or** e_2	$\mathcal{C}(e_1) + (1 - \mathcal{P}(e_1)) \cdot \mathcal{C}(e_2)$	COR	1	SOR
e_1 **and** e_2	$\mathcal{C}(e_1) + \mathcal{P}(e_1) \cdot \mathcal{C}(e_2)$	CAND	1	SAND
some $\$x$ **in** s **satisfies** $p(\$x)$	$\mathcal{C}(s) + \mathcal{S}(s) \cdot \mathcal{C}(p)$	CNSOME	1	SSOME
	$\mathcal{C}(s) + \frac{1-(1-\mathcal{P}(p))^{\mathcal{S}(s)}}{\mathcal{P}(p)} \mathcal{C}(p)$	CSOME		
every $\$x$ **in** s **satisfies** $p(\$x)$	$\mathcal{C}(s) + \frac{1-\mathcal{P}(p)^{\mathcal{S}(s)}}{1-\mathcal{P}(p)} \mathcal{C}(p)$	CEVERY	1	SEVERY
$e_1[e_2]$	$\mathcal{C}(e_1) + \mathcal{S}(e_1) \cdot \mathcal{C}(e_2)$	CPRED	$\mathcal{P}(e_2) \cdot \mathcal{S}(e_1)$	SPRED
$QName(e_1, e_2, \ldots, e_N)$	$\mathcal{C}(e_1) + \mathcal{C}(e_2) + \cdots + \mathcal{C}(e_N)$ $+\mathcal{C}(\text{function\_body}(QName))$	CFCALL	$\mathcal{S}(\text{function\_body}(QName))$	SFCALL

expression	probability	label		
some $\$x$ **in** s **satisfies** $p(\$x)$	$1 - (1 - \mathcal{P}(p))^{\mathcal{S}(q)}$	PSOME		
every $\$x$ **in** s **satisfies** $p(\$x)$	$\mathcal{P}(p)^{\mathcal{S}(q)}$	PEVERY		
e_1 **and** e_2	$\mathcal{P}(e_1) \cdot \mathcal{P}(e_2)$	PAND		
e_1 **or** e_2	$1 - (1 - \mathcal{P}(e_1)) \cdot (1 - \mathcal{P}(e_2))$	POR		

Table 1. Summary of cost functions