Containment of XPath expressions: an Inference and Rewriting based approach

Jean-Yves Vion-Dury and Nabil Layaïda

June 19, 2003

Abstract

XPath is simple query language for XML documents which allows navigating in XML trees and returning a set of matching nodes. It is used in XML Schema to define keys and in XLink and XPointer to reference portions of documents. XPath is a fundamental part of the XSLT and XQuery languages as it allows to define matching expressions for patterns and provides node selectors to filter elements in the transformations.

We propose to study the containment and equivalence of XPath expressions using an inference system combined with a rewriting system. The inference system allows to assert and prove properties on a class of expressions. In order to keep the proof system compact, we propose a re-writing architecture which allows to transform remaining expressions in a disjunctive normal form compatible with this class. In contrast with model based approaches, the inference and rewriting systems are applied to the XPath language directly. We believe this will help understanding the underlying issues of deciding containment on the language itself.

1 XPath Containment and Equivalence

XPath is a simple yet powerful query language for XML documents, which allows navigating in XML trees and returning a set of matching nodes. It is used in XML Schema to define keys and in XLink and XPointer to reference portions of documents. More notably, it represents a fundamental part of the XSLT and XQuery languages, as it allows to define matching expressions for patterns and provides node selectors to filter elements in the transformations.

For a given XPath expression p and an XML input tree t, $p(t)$ denotes the set of nodes from t returned by the evaluation of p. More precisely, an expression p is evaluated relatively to a particular node of t, called the context node (noted x). Thus the selection of a node set by p in a tree t with respect to a context node x of t is written $p^x(t)$. The containment relation between two XPath expressions p_1 and p_2 (denoted $p_1 \leq p_2$) holds true when, for any XML tree t and any context node x, the set of nodes selected by $p_1^x(t)$ is included in the set selected by $p_2^x(t)$.

$$\forall t, \forall x \in t, \quad p_1 \leq p_2 \text{ iff } p_1^x(t) \subseteq p_2^x(t)$$

It is worth noting that the equivalence relation between two expressions (denoted $p_1 \equiv p_2$) can be expressed using two containment relations. (i.e. $p_1 \leq p_2$ and $p_2 \leq p_1$), and given a suitable algorithm, equivalence is reducible in polynomial time to containment [5].

The containment and equivalence relations for XPath expressions are essential in each of the previously mentioned areas. In addition, several fundamental problems reduce directly to containment or equivalence, such as expression optimization and keys inference. For example, if an expression p_1 represents a key for a given Schema, then all p expressions such that $(p \leq p_1)$ are also keys.

Containment is also a key component for the static analysis of XSLT Transformations. It can reveal two

aspects of the expressions valuable for transformation designers and query programmers : consistency and performance. In practice, complex XPath expressions turns out to be difficult to interpret, therefore errors can be easily introduced. The consistency of an expression p can be verified by checking if it is contained in the empty path [1] ($p \leq \perp$). Performance issues can also be easily understood with the following example: given a path $p \equiv p_1|p_2$, if we have $p_1 \leq p_2$, then evaluating p_1 is redundant since all nodes have already been selected in the evaluation of p_2. Therefore, re-formulating p by the equivalent expression p_2 represents an optimization of the evaluation of p. Query optimization aims at reducing unnecessary operations such as tree navigation in the evaluation and can be more complex to determine than in the previous example.

2 Related Work

In [5], the authors focus on a complexity analysis of the containment problem, based on a rudimentary fragment of XPath. The considered subset is based on four constructs, namely : the child axis (/), descendant axis (//), wild-cards (*) and qualifiers ([]). XPath expressions are then modeled as tree patterns. The containment problem is then (re)formulated in terms of tree pattern homomorphism search. This search is used to determine the complexity of the containment for a range of languages which are subsets of the four constructs.

In [2], the containment is studied using DTD information translated into Simple XPath Integrity Constraints (SXIC). The paper shows that for particular SXIC constraints, namely unbounded SXICs, the containment problem is undecidable. For the general problem, it indicates that the decidability of the containment for full-fledged XPath remains an open question. Neven and Schwentick [3] established recently that containment under DTD constraints is in fact undecidable. If this is certainly a significant outcome, the proof relies on a somehow complex reduction to the Post Correspondence Problem (PCP [4]), where the basic tree linearization function **yield** plays a key role ; unfortunately, it brings in our opinion insufficient help in understanding the deep nature of the containment problem.

In [1], the work examines the symmetry of most of XPath axes in order to optimize XML database queries. An equivalence of location paths involving "reverse axes" is first established. This equivalence relation is used to re-write XPath expressions into reverse axes free equivalents which reduces the cost of the evaluation process. The paper shows that most of the "duplicate" axes can be removed and that for some others an equivalent formulation in the XQuery language is possible.

The work found in these studies can be split in two broad categories:

1. *Model based approaches.* It relies on a modeling of the expressions as tree patterns or tree automatons. Containment is then handled in terms of low level operations on these structures.

2. *Syntax-oriented approaches.* In contrast, the latter approaches applies syntactic (re-writing) operations on expressions or containment assertions directly.

Syntax-oriented approaches have several advantages. First, rewritten expressions are XPath expressions and can be interpreted according to the same common semantics. This provides more readability of the different steps used to state the containment.

Second, since containment is established on XPath expressions directly, it does not introduce different properties such as those introduced by the models. For instance, tree patterns do not capture the ordering of nodes in trees despite the fact that this feature is important for XPath expressions.

Finally, reasoning on automatons for containment is achieved by applying automaton transformations (complement, intersection, determination). This approach suffers from the lack of reversible methods which allows to translate the results of these operations back from automatons to path expressions. This precludes for instance, application such as optimizing an XSLT stylesheet by static analysis and

[1] the empty path $\perp$ selects always an empty node set. First introduced in [1], - up to our knowledge - in order to ease formal treatments

relevant replacement of xpath expressions.

This paper is also an attempt to address most of the issues related to the syntax-oriented approach: mastering the combinatorial complexity and preserving the scalability of the formal tools.

3 Goal of the paper

The goal of this paper is twofold. First, it defines a formal method which aims to analyze and reason about XPath expressions thank to (*i*) an inference system, combined to (*ii*) a rewriting architecture.

We believe that the formalization we propose is applicable to the entire XPath language and represents a generic tool for the study of the related problems. Additionally, this mathematical tool is relevant to the study of other aspects such as optimization and path simplification since it helps exploring, asserting and proving general properties on the XPath language. In our knowledge, this is the first attempt in defining such a logical architecture dedicated to the XPath language.

Building a tractable inference system is not easy when dealing with such a powerful language, especially considering the combinatorial way to express equivalent paths. The re-writing system's goal is to produce a normal form for XPath expressions that eases establishing containment while maintaining reasonably small the inference system.

The second goal of the paper is to analyze XPath in order to prove containment assertions on a large sample of the language. In the light of this analysis, we look forward to better understanding the language constructs sources of high computational complexity and decidability problems, as reported in the previously mentioned studies. In the longer run, we are paving the way to the study of important properties such as soundness and completeness using the containment approach we propose.

The rest of the paper is organized as follows. In section 4, we present a formal syntax and semantics for XPath, which we think captures faithfully a significant fragment of XPath 2.0 [6], including all axes and some of the newly proposed operators such as the **for** and the **if** $\cdots$ **then** $\cdots$ **else** constructs. Section 5 presents a proof system for containment and equivalence assertions. This inference system is then associated with a transformation architecture, presented in section 6. After concluding, we draw some perspectives on this ongoing work, including indications on the mathematical characterization of the proposed approach.

4 XPath Syntax and Semantics

We propose a restricted but faithful version of XPath 2.0 [6], a subset and/or a combination of the ones proposed by Wadler ([7, 8]), and Olteanu & al. ([1]), plus some extensions that ease formal analysis, such as the explicit root node $\wedge$, the void path $\perp$ (proposed and defined in [1]). It is one of our future goals to extend this subset toward even more realistic extents, notably including important functions such as **position**() in qualifiers, as in [8].

Our underlying goal is to remain close to the engineering reality that many developers have to deal with, and thus trying to give the best applicative field to the present work.

4.1 Syntax

The syntax is defined in two stages: *(i)* the core language, and *(ii)* the syntactic sugars, which just rewrite into expressions of the core.

$$
\begin{array}{rcl}
p & ::= & \wedge \;\mid\; \perp \;\mid\; p \,|\, p \;\mid\; \mathrm{a}::N \;\mid \\
 & & (p) \;\mid\; p/p \;\mid\; p[q] \;\mid \\
 & & \$v \;\mid\; \textbf{for } \$v \textbf{ in } p \textbf{ return } p
\end{array}
$$

Note that according to XPath 2.0 specification, and to proposal [1], but not in accordance with the proposal of [7], the axes $a :: N$ cannot contain other paths (e.g. $a :: (p_1|p_2)$). a node test N, is either an unqualified name n, the wildcard symbol $*$ or a test function among **text**(), **node**(), **comment**(), **processing-instruction**(), **element**() ; when N is under the $ns : n$ form, ns is considered as a namespace prefix in accordance to the specification, and processed accordingly. More precisely, it is rewritten into a namespace attribute (see rule r_{5f}, fig. 4).

The important extension we propose with respect to qualifiers is the node set inclusion constraint $p \sqsubseteq p_2$, which brings extra expressive power and interesting possibilities for containment inference.

$$q ::= \textbf{true} \mid \textbf{false} \mid (q) \mid \textbf{not } q \mid q \textbf{ and } q \mid q \textbf{ or } q \mid p \sqsubseteq p$$

A (partial) reflexive ordering of N elements is defined in figure 3. The a symbol denotes *axes*, ranging over the whole set defined in the W3C specification

$$\begin{aligned} a \in & \{\textit{self, attribute, namespace}\} \\ \cup & \{\textit{child, parent}\} \\ \cup & \{\textit{descendant, ancestor}\} \\ \cup & \{\textit{descendant-or-self, ancestor-or-self}\} \\ \cup & \{\textit{following, preceding,}\} \\ \cup & \{\textit{preceding-sibling, following-sibling}\} \end{aligned}$$

We define the syntactic sugars (see fig. 4 for their translation into core expressions)

$$p ::= p//p \mid .. \mid . \mid N \mid @N \mid ns:N \mid \textbf{if } q \textbf{ then } p \textbf{ else } p$$

and similarly for qualifiers (see figure 5)

$$q ::= p \mid p == p \mid p \neq p \mid p \sqsubset p \mid p \sqsupset p \mid p \sqsupseteq p$$

We consider that the / and | operators are fully associative, and that the precedence ordering is (from the tightest to the loosest):

$$\begin{aligned} & \mathrm{a}::N < p[q] < p/p < \\ & \textbf{for } \$v \textbf{ in } p \textbf{ return } p < p|p \end{aligned}$$

so that for instance

$$\wedge/\mathit{child}::n[q]/\mathit{child}::n_2[q'] \mid \wedge/\mathit{descendant}::*$$

is syntactically understood as

$$\wedge/(\mathit{child}::n[q])/(\mathit{child}::n_2[q']) \mid \wedge/(\mathit{descendant}::*)$$

The precedence of boolean operators is the standard one (**not** < **and** < **or**).

4.2 Denotational Semantics

We reused the denotational definition of [1], also inspired from [7], extended with an explicit document root $\wedge$ and with an execution context ϕ that uniquely associates variable names to tree nodes.

Document model. The XPath semantics relies on a document-as-a-tree model. A linear ("flat") document is seen as a well-formed tree after a successful parsing. A tree is modeled as a set of "typed" nodes (*element, text, comment, processing-instruction, attribute, namespace*, and *root*; the type of a node can be checked by a corresponding unary predicate).

A well-formed tree contains only one root node, which has no parent, no attribute and no namespace but may have any other kind of nodes as children. Moreover, only elements can have children. Nodes, identified by x and x_i in the sequel, are fully connected in order to form a tree [2]. This structural property relies on the *parent/child* relation $\rightarrowtriangle$ that characterizes the tree, and also its transitive closure $\rightarrowtriangle^+$. Moreover, a strict ordering, the document ordering $\ll$, is defined on every node x of a tree t, and reflect the order of tag occurrence in the linear document. The ordering relation $x_1 \ll x_2$ is **true** if the opening tag of x_1 appears strictly before the opening tag of x_2 in the document, **false** otherwise (thus, ancestors of a node x are considered as being before x; that is, nodes are totally ordered).

Semantics of selection. The selection is defined relatively to a context node x, and the execution context ϕ. If $\phi = \{\cdots, v = x, \cdots\}$ then $\phi(v) = x$. The function $\mathcal{S}$ is inductively defined, i.e. uses itself for its own definition. The induction is even double, since it relies on the $\mathcal{Q}$ function (defines qualifiers, presented in the next paragraph) which itself uses $\mathcal{S}$. This is however quite common in denotational semantics, and just reflects faithfully the syntactic structure

[2] Every node x of the tree is reachable from the root ; mathematically, $\mathrm{root}(x) \rightarrowtriangle^+ x$

of the XPath language.

$$\mathcal{S} : \text{Pattern} \longrightarrow \text{Env} \longrightarrow \text{Node} \longrightarrow \text{Set(Node)}$$

$$\begin{array}{lcl}
\mathcal{S}[\![\wedge]\!]^{\phi}_{x} & = & \{x_1 \mid x_1 \twoheadrightarrow^{+} x \wedge \text{root}(x_1)\} \\
\mathcal{S}[\![\bot]\!]^{\phi}_{x} & = & \emptyset \\
\mathcal{S}[\![(p)]\!]^{\phi}_{x} & = & \mathcal{S}[\![p]\!]^{\phi}_{x} \\
\mathcal{S}[\![p_1|p_2]\!]^{\phi}_{x} & = & \mathcal{S}[\![p_1]\!]^{\phi}_{x} \cup \mathcal{S}[\![p_2]\!]^{\phi}_{x} \\
\mathcal{S}[\![p_1/p_2]\!]^{\phi}_{x} & = & \{x_2 \mid x_1 \in \mathcal{S}[\![p_1]\!]^{\phi}_{x} \wedge x_2 \in \mathcal{S}[\![p_2]\!]^{\phi}_{x_1}\} \\
\mathcal{S}[\![p[q]]\!]^{\phi}_{x} & = & \{x_1 \mid x_1 \in \mathcal{S}[\![p]\!]^{\phi}_{x} \wedge \mathcal{Q}[\![q]\!]^{\phi}_{x_1}\} \\
\mathcal{S}[\![\$v]\!]^{\phi}_{x} & = & \{\phi(v)\} \\
\mathcal{S}[\![a::N]\!]^{\phi}_{x} & = & \{x_1 \mid f_a(x) \wedge \mathcal{T}_a(x_1, N)\}
\end{array}$$

The notation f_a corresponds to the functions defined in the table below.

axis (a)	$f_a(x)$
self	$\{x\}$
child	$\{x_1 \mid x \twoheadrightarrow x_1\}$
parent	$\{x_1 \mid x_1 \twoheadrightarrow x\}$
descendant	$\{x_1 \mid x \twoheadrightarrow^{+} x_1\}$
ancestor	$\{x_1 \mid x_1 \twoheadrightarrow^{+} x\}$
descendant-or-self	$\{x\} \cup \{x_1 \mid x \twoheadrightarrow^{+} x_1\}$
ancestor-or-self	$\{x\} \cup \{x_1 \mid x_1 \twoheadrightarrow^{+} x\}$
following-sibling	$\text{sibling}(x) \cap \text{following}(x)$
preceding-sibling	$\text{sibling}(x) \cap \text{preceding}(x)$
preceding	$\{x_1 \mid x_1 \ll x\}$
following	$\{x_1 \mid x \ll x_1\}$
attribute	$\{x_1 \mid x \twoheadrightarrow x_1 \wedge \text{attribute}(x_1)\}$
namespace	$\{x_1 \mid x \twoheadrightarrow x_1 \wedge \text{namespace}(x_1)\}$
sibling	$\{x_2 \mid x_1 \twoheadrightarrow x \ \wedge \ x_1 \twoheadrightarrow x_2\}$

As noticed in the XPath 2.0 specification, the axes *self, ancestor, descendant, following* and *preceding* constitute a (non-overlapping) partition of the tree.

The $\mathcal{T}$ function performs a node test, according to the table below

N	a	$\mathcal{T}_a(N,x)$
n		name(x)=n
*	*attribute*	attribute(x)
*	*namespace*	namespace(x)
*	other	element(x)
text()		text(x)
comment()		comment(x)
processing-instruction()		pi(x)
element()		element(x)
node()		**true**

The **for** expression is the only one that introduces a new (variable, node value) pair into the context

$$\begin{array}{lcl}
& & \mathcal{S}[\![\textbf{for } \$v \textbf{ in } p_1 \textbf{ return } p_2]\!]^{\phi}_{x} \\
& = & \{x_2 \mid x_1 \in \mathcal{S}[\![p_1]\!]^{\phi}_{x} \wedge x_2 \in \mathcal{S}[\![p_2]\!]^{\phi,v=x_1}_{x}\}
\end{array}$$

semantics of qualifiers. The originality here is in the inclusion test (last line), directly expressed through a set-theoretic inclusion of selection sets.

$$\mathcal{Q} : \text{Qualifier} \longrightarrow \text{Env} \longrightarrow \text{Node} \longrightarrow \text{Boolean}$$

$$\begin{array}{lcl}
\mathcal{Q}[\![\textbf{true}]\!]^{\phi}_{x} & = & \textbf{true} \\
\mathcal{Q}[\![\textbf{false}]\!]^{\phi}_{x} & = & \textbf{false} \\
\mathcal{Q}[\![q_1 \textbf{ and } q_2]\!]^{\phi}_{x} & = & \mathcal{Q}[\![q_1]\!]^{\phi}_{x} \wedge \mathcal{Q}[\![q_2]\!]^{\phi}_{x} \\
\mathcal{Q}[\![q_1 \textbf{ or } q_2]\!]^{\phi}_{x} & = & \mathcal{Q}[\![q_1]\!]^{\phi}_{x} \vee \mathcal{Q}[\![q_2]\!]^{\phi}_{x} \\
\mathcal{Q}[\![(q)]\!]^{\phi}_{x} & = & \mathcal{Q}[\![q]\!]^{\phi}_{x} \\
\mathcal{Q}[\![\textbf{not } q]\!]^{\phi}_{x} & = & \neg\mathcal{Q}[\![q]\!]^{\phi}_{x} \\
\mathcal{Q}[\![p_1 \sqsubseteq p_2]\!]^{\phi}_{x} & = & \mathcal{S}[\![p_1]\!]^{\phi}_{x} \subseteq \mathcal{S}[\![p_2]\!]^{\phi}_{x}
\end{array}$$

5 Inferring containment

5.1 Judgments, context and logical rules

The inference system we present in this paper, noted $\mathcal{I}$ allows asserting and proving containment properties by using judgments, noted $\gamma \ \vdash \ p_1 \leq p_2$ and rules like

$$\frac{A_1 \cdots A_n}{B}\ [r]$$

where A_i and B are judgments, and r is the rule name. Rules like these mean that if all judgments A_i are true, then B is true. This well-known notation allows to build proofs as a tree such as

$$\dfrac{\dfrac{A \quad B}{C}\ [r_1] \qquad \dfrac{D \quad E}{F}\ [r_2]}{G}\ [r_1]$$

provided the user is able to unify expressions with the logical constituent found in rules. Judgments like $\gamma \ \vdash \ p_1 \leq p_2$ means that the path p_1 is contained by the path p_2, given the context γ. This latter contains couples of variable names and values, noted $v{:}p$, where variables names v are uniquely defined. The context is also a mapping, and $\gamma(v)$ will return p, provided v is defined in γ, abbreviated as

$v \in \gamma$. Sometimes, the context may be omitted from the proof tree, when it is invariant or understood as not significant. The assertion "the path p_1 is contained in the path p_2" means intuitively that the set of nodes selected by p_1 is included [3] in the set of nodes selected by p_2.

Proof trees can also be developed by using an equivalence relation $\equiv$ on xpath expressions or judgments. If for instance, we define a path $P_1 \equiv P_2/c$, it is legal by construction [4] to derive a proof step by simple substitution, such as [5]

$$\dfrac{\dfrac{\cdots}{P_2/c \leq p}\ [r_1]}{P_1 \leq p}\ [\equiv]$$

The figures 6 and 7 propose useful equivalences, which are provable using the formal semantics proposed in section 4.2 (see [1] for similar demonstrations). We note $\mathcal{I}^{\equiv}$ the inference system $\mathcal{I}$, defined in the coming sub-section, used together with the equivalence relation.

5.2 Basic containments

We introduce first two basic rules expressing that, in whatever context γ, it is always true that *(i)* the void path $\perp$ is contained in whatever other path p, and *(ii)* any path p is contained in itself

$$\dfrac{}{\gamma \vdash \perp \leq p}\ [c_1] \qquad \dfrac{}{\gamma \vdash p \leq p}\ [c_2]$$

The union operator can be present either at the left hand or at the right hand of a containment assertion. The latter case raises two possible choices for further containments

$$\dfrac{\gamma \vdash p \leq p_1}{\gamma \vdash p \leq p_1 \mid p_2}\ [c_{2a}] \qquad \dfrac{\gamma \vdash p \leq p_2}{\gamma \vdash p \leq p_1 \mid p_2}\ [c_{2b}]$$

while when located at the left hand, it requires both left sub-terms to be contained by the right term, as expressed by

$$\dfrac{\gamma \vdash p_1 \leq p \quad \gamma \vdash p_2 \leq p}{\gamma \vdash p_1 \mid p_2 \leq p}\ [c_{2c}]$$

Now we can prove that $a/b \leq a/b \mid c/d$ in the empty context thank to the following proof tree

$$\dfrac{\dfrac{}{\emptyset \vdash a/b \leq a/b}\ [c_2]}{\emptyset \vdash a/b \leq a/b \mid c/d}\ [c_{2a}]$$

Such proof relies on a very basic mechanism, the unification, which allows to identify the term $a/b \leq a/b \mid c/d$ to the term $p \leq p_1 \mid p_2$ in the bottom of the rule c_{2a}, thanks to a substitution

$$\begin{array}{lcl} p & \equiv & a/b \\ p_1 & \equiv & a/b \\ p_2 & \equiv & c/d \end{array}$$

Applying this substitution to the top of the rule allows the further development of the tree.

Finer comparisons require handling the / composition operator, quite fundamental in XPath. This is achieved by this very generic rule that conducts symmetrical comparisons

$$\dfrac{\gamma \vdash p_1 \leq p_3 \quad \gamma \vdash p_2 \leq p_4}{\gamma \vdash p_1/p_2 \leq p_3/p_4}\ [d_2]$$

and using $[d_2]$, we can prove that $a/b \leq (a|c)/b$:

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash a \leq a}\ [c_2]}{\emptyset \vdash a \leq a|c}\ [c_{2a}] \quad \dfrac{}{\emptyset \vdash b \leq b}\ [c_2]}{\emptyset \vdash a/b \leq (a|c)/b}\ [d_2]$$

The context is useful to handle paths defined with variables. Using rules below

$$\dfrac{\gamma \vdash \gamma(v) \leq p}{\gamma \vdash \$v \leq p}\ [d_{3a}] \qquad \dfrac{\gamma \vdash p \leq \gamma(v)}{\gamma \vdash p \leq \$v}\ [d_{3b}]$$

one can prove that $\$v/b \leq (a|c)/b$ in a context $\gamma = v{:}c|a$

$$\dfrac{\dfrac{\dfrac{}{\gamma \vdash c|a \leq a|c}\ [t_1]}{\gamma \vdash \$v \leq a|c}\ [d_{3a}] \quad \dfrac{}{\gamma \vdash b \leq b}\ [c_2]}{\gamma \vdash \$v/b \leq (a|c)/b}\ [d_2]$$

[3] following the set-theoretic definition of inclusion

[4] mathematically, this equivalence is necessarily fully congruent with respect to a sound containment relation

[5] In proof tree, $[\equiv]$ are usually not shown for the sake of compactness

Note that t_1 is not a basic rule but the following theorem (the context, invariant, is omitted)

$$\dfrac{\dfrac{\dfrac{}{p_1 \leq p_1}\,[c_2]}{p_1 \leq p_2|p_1}\,[c_{2b}] \quad \dfrac{\dfrac{}{p_2 \leq p_2}\,[c_2]}{p_2 \leq p_2|p_1}\,[c_{2a}]}{p_1|p_2 \leq p_2|p_1}\,[c_{2c}]$$

We need more rules in order to compare steps. Let us consider the following rule, where a, a' denote axis names and N, N' node tests:

$$\dfrac{\mathrm{a} \leq \mathrm{a'} \quad N \leq N'}{\gamma \vdash \mathrm{a}::N \leq \mathrm{a'}::N'}\,[d_1]$$

Considering that axes are partially ordered through the reflexive relation of Fig. 2, and that node tests are similarly ordered by the relation of Fig. 3 , we can prove that $child::b$ is contained in $descendant::*$, since

$$\dfrac{child \leq descendant \quad b \leq *}{child::b \leq descendant::*}\,[d_1]$$

5.3 Qualifiers

Comparing qualified paths such as $p[q]$ requires a new judgment, the implication. This is captured by the following rule

$$\dfrac{\gamma \vdash p_1 \leq p_2 \quad \gamma \vdash q_1 \Rightarrow q_2}{\gamma \vdash p_1[q_1] \leq p_2[q_2]}\,[d_3]$$

Consider for instance $a[b/c] \leq a[*]$, which should be provable, since both expression conditionally select a elements, but with more restrictive condition for the former one. In our approach, it is captured by $b/c \Rightarrow *$, which is equivalent by definition to **not** $(b/c \sqsubseteq \bot) \Rightarrow$ **not** $(* \sqsubseteq \bot)$. The rules for reasoning on this kind of relations are developed in a coming subsection.

We focus now on applying d_3 in combination with equivalences of figure 7 in order to illustrate how to assert various containment properties involving qualifiers. For instance, $[d_3]$ allows proving $a[b][c] \leq a[*]$, if we admit for now the sub-proof related to the implication:

$$\dfrac{\dfrac{\dfrac{}{a \leq a}\,[c_2] \quad \dfrac{\cdots}{b \textbf{ and } c \Rightarrow *}\,[\cdots]}{a[b \textbf{ and } c] \leq a[*]}\,[d_4]}{a[b][c] \leq a[*]}$$

It is similarly possible to compare a qualified xpath with an unqualified expression, e.g. $a[b] \leq a|b$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a \leq a}\,[c_2] \quad \dfrac{\cdots}{b \Rightarrow \textbf{true}}\,[\cdots]}{a[b] \leq a[\textbf{true}]}\,[d_3]}{a[b] \leq a}}{a[b] \leq a|b}\,[c_{2a}]$$

5.4 The for construct

the **for** construct must also be compared too, and we consider first the left hand case

$$v \notin \gamma \quad \dfrac{\gamma, v{:}p_1 \vdash p_2 \leq p}{\gamma \vdash \textbf{for } \$v \textbf{ in } p_1 \textbf{ return } p_2 \quad \leq \quad p}\,[d_{4a}]$$

Note that the content of the **for** construct is evaluated in a new enriched context, where the variable $\$v$ has been introduced with the right value p_1. Here, the variable name must not be already defined in the context.

The right hand case is pretty similar, excepted that an additional condition must be verified, in order to preserve soundness. If p_1 is empty in the expression **for** $\$v$ **in** p_1 **return** p_2, the variable $\$v$ won't take any value, and thus the inner expression p_2 will never be evaluated. In conformance with the formal semantics, the whole construct will select nothing, and thus can only contain the $\bot$ path. For instance, it's easy to prove the a/b is equivalent to **for** $\$v$ **in** a **return** $\$v/b$, but this is not the case for b and **for** $\$v$ **in** a **return** b. Indeed, this latter path is only equivalent to $b[a]$.

$$v \notin \gamma \quad \dfrac{\begin{array}{c}\gamma, v{:}p_1 \vdash p \leq p_2 \\ \gamma \vdash p_1 \sqsubseteq \bot \Rightarrow p \sqsubseteq \bot\end{array}}{\gamma \vdash p \quad \leq \quad \textbf{for } \$v \textbf{ in } p_1 \textbf{ return } p_2}\,[d_{4b}]$$

Name conflicts occur when comparing two different **for** constructs which define the same variable, since

the inference system only use one context. Let us consider the following example

$$\begin{cases} P_1 & = & \textbf{for } \$v \textbf{ in } a \textbf{ return } (\$v/b) \\ P_2 & = & \textbf{for } \$v \textbf{ in } a/b \textbf{ return } (\$v/*) \\ P_1/c & \equiv & P_2 \end{cases}$$

In such case, it's possible to use an α-substitution, i.e. a renaming operation propagated throughout the whole term, very much like in the lambda calculus framework. The substitution of $\$v$ by $\$w$ in path p is noted $p[\$w/\$v]$ and we do have $p \equiv p[\$w/\$v]$ as long as $\$w$ is not used in p. This latter condition is required prior to using any α-substitution. Thus the proof of $P_1/c \leq P_2$ comes in four parts, only three of them being developed here (the forth one, the C judgment (an implication) will be shown later)

$$\dfrac{A \quad \dfrac{\dfrac{\dfrac{}{child \leq child} \quad \dfrac{}{c \leq *}}{child :: c \leq child :: *}\,[d_1]}{c \leq *}\,[d_2] \quad C}{v : a/b \;\vdash\; P_1/c \leq \$v/*}\,[d_{4b}]}{\emptyset \;\vdash\; P_1/c \leq P_2}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{B \quad \dfrac{}{b \leq b}\,[c_2]}{v : a/b, w : a \;\vdash\; \$w/b \leq a/b}\,[d_2]}{v : a/b, w : a \;\vdash\; \$w/b \leq \$v}\,[d_{3b}]}{v : a/b \;\vdash\; \textbf{for } \$w \textbf{ in } a \textbf{ return } \$w/b \leq \$v}\,[d_{4a}]}{v : a/b \;\vdash\; P_1[\$w/\$v] \leq \$v}}{v : a/b \;\vdash\; P_1 \leq \$v}}{A}$$

$$\dfrac{\dfrac{\dfrac{}{v : a/b, w : a \;\vdash\; a \leq a}\,[c_2]}{v : a/b, w : a \;\vdash\; \$w \leq a}\,[d_{3a}]}{B}$$

$$\dfrac{\emptyset \;\vdash\; a/b \sqsubseteq \bot \Rightarrow P_1/c \sqsubseteq \bot}{C}$$

5.5 Qualifier implication

Now we present the second piece of our inference system, based on an implication judgment $\gamma \;\vdash\; q_1 \Rightarrow q_2$, which means if q_1 is true in context γ, then q_2 is true in the same context. We propose first the most general set of rules (the context γ, invariant, is omitted for clarity).

$$\dfrac{p_3 \sqsubseteq p_1 \quad p_2 \sqsubseteq p_4}{p_1 \sqsubseteq p_2 \Rightarrow p_3 \sqsubseteq p_4}\,[e_1]$$

This first rule permits many trivial (but often useful) inferences, such as $p_1 \sqsubseteq p_2 \Rightarrow p_1 \sqsubseteq p_2$ and also less obvious ones like $descendant :: * \sqsubseteq b \Rightarrow */b \sqsubseteq *$.

All other e_i rules defined hereafter are directly logically derived from the standard boolean implication.

$$\dfrac{q_2 \Rightarrow q_1}{\textbf{not } q_1 \Rightarrow \textbf{not } q_2}\,[e_2] \qquad \dfrac{}{q \Rightarrow \textbf{true}}\,[e_3 a] \qquad \dfrac{}{\textbf{false} \Rightarrow q}\,[e_3 b]$$

The following subset deals with conjunction

$$\dfrac{q_2 \Rightarrow q}{\begin{array}{l} q_1 \textbf{ and } q_2 \Rightarrow q \\ q_2 \textbf{ and } q_1 \Rightarrow q \end{array}}\,[e_4] \qquad \dfrac{q \Rightarrow q_1 \quad q \Rightarrow q_2}{q \Rightarrow q_1 \textbf{ and } q_2}\,[e_5]$$

While this one addresses disjunction

$$\dfrac{q \Rightarrow q_2}{\begin{array}{l} q \Rightarrow q_1 \textbf{ or } q_2 \\ q \Rightarrow q_2 \textbf{ or } q_1 \end{array}}\,[e_6] \qquad \dfrac{q_1 \Rightarrow q \quad q_2 \Rightarrow q}{q_1 \textbf{ or } q_2 \Rightarrow q}\,[e_7]$$

5.6 Inferring emptiness

XPath Emptiness, e.g. $a/b \sqsubseteq \bot$ appears quite often in qualifiers. It turned out that such relation represent a particular containment case we have to deal with. An important qualifier equivalence, defined below, characterizes this particularity.

$$p_1/p_2 \sqsubseteq \bot \quad \equiv \quad p_1[\textbf{not } p_2 \sqsubseteq \bot] \sqsubseteq \bot$$

Used in conjunction with f_1 or f_2 below, it opens many possibilities for building proofs

$$\dfrac{p_1 \sqsubseteq \bot \Rightarrow p_2 \sqsubseteq \bot}{p_1[q_1] \sqsubseteq \bot \Rightarrow p_2[q_2] \sqsubseteq \bot}\,[f_2]$$

$$\dfrac{p_1 \leq p_2 \quad \textbf{not } q_1 \Rightarrow \textbf{not } q_2}{p_1[q_1] \sqsubseteq \bot \Rightarrow p_2[q_2] \sqsubseteq \bot}\,[f_3]$$

It can also be used in order to prove the previous C assertion

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a/b \leq a/b}\,[c_2] \quad \dfrac{}{\bot \leq \bot}\,[c_1]}{a/b \sqsubseteq \bot \Rightarrow a/b \sqsubseteq \bot}\,[e_1]}{a/b \sqsubseteq \bot \Rightarrow P_1 \sqsubseteq \bot}}{a/b[\mathbf{true}] \sqsubseteq \bot \Rightarrow P_1[\mathbf{not}\ c \sqsubseteq \bot] \sqsubseteq \bot}\,[f_2]}{a/b \sqsubseteq \bot \Rightarrow P_1/c \sqsubseteq \bot}}{C}$$

Rules e_i, f_2, f_3 still cannot address two particular cases. this first one is captured by the following rule

$$\dfrac{}{child::* \sqsubseteq \bot \Rightarrow descendant::N \sqsubseteq \bot}\,[f_4]$$

Intuitively, $descendant::N$ for instance can be understood as

$$\underbrace{child::*/\cdots/child::N}_{n \geq 0}$$

On such a path, one could apply the rule f_2, e_1, if we were allowed to handle infinite terms. The other similar case to be considered is

$$\dfrac{}{parent::* \sqsubseteq \bot \Rightarrow ancestor::N \sqsubseteq \bot}\,[f_5]$$

Other "multi-steps" axes, such as *following*, have no single step equivalent, thus they are not to be considered.

5.7 Multi-step axis

At this point, our inference system doesn't capture an important containment case, such as

$$*/b/b \leq descendant::b$$

The reader may verify that no rule can be successfully applied in order to complete the tree

$$\dfrac{\dfrac{\dfrac{?}{*/b \leq .}\,[?] \quad \dfrac{\dfrac{child \leq descendant \quad b \leq b}{child::b \leq descendant::b}\,[d_1]}{b \leq descendant::b}}{*/b/b \leq ./descendant::b}\,[d_2]}{*/b/b \leq descendant::b}$$

We propose now a quite generic rule, since it addresses all "multi-step" axes

$$a_2 \in \{descendant, ancestor, following, preceding, following\text{-}sibling, preceding\text{-}sibling\}$$

$$\dfrac{a_1::N_1 \leq a_2::N_2 \quad p_1 \leq p_2/a_2::*}{p_1/a_1::N_1 \leq p_2/a_2::N_2}\,[g_1]$$

With g_1, it's possible to prove:

$$\dfrac{\dfrac{child \leq desc \quad b \leq b}{child::b \leq desc::b}\,[d_1] \quad \dfrac{\dfrac{[c_2]}{a \leq a} \quad D}{a/desc::b \leq a/desc::*}\,[d_2]}{a/desc::b/b \leq a/desc::b}\,[g_1]$$

with

$$\dfrac{\dfrac{desc \leq desc \quad b \leq *}{desc::b \leq desc::*}\,[d_1]}{D}$$

5.8 Pushing context in expressions

Now comes much more difficult cases. Our inference system cannot establish two important cases captured by the examples below

$$\begin{array}{rcrcl} E & \equiv & a/b/c & \leq & a[desc::c]/*/* \\ F & \equiv & a/b & \leq & */*[anc::a] \end{array}$$

The problem expressed by the E judgment is that the right context b/c is missing in order to compare the a node from the $a/b/c$ expression with the a node from the second one. Using d_2, the proof construction attempt is stopped after applying d_3:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a \leq a}\,[c_2] \quad \dfrac{?}{\mathbf{true} \Rightarrow desc::c}\,[?]}{a[\mathbf{true}] \leq a[desc::c]}\,[d_3]}{a \leq a[desc::c]} \quad \dfrac{\cdots}{b/c \leq */*}\,[d_2]}{a/b/c \leq a[desc::c]/*/*}\,[d_2]$$

However, since c is present as a second level child of a in $a/b/c$, $a \leq a[desc::c]$ should be deducible since the right hand qualifier condition is actually satisfied.

Similar situation holds for the F judgment; the problem comes from the left context a/b of the c element comparison. It should be possible to deduce from this context that c has indeed a a ancestor.

We propose to tackle this problem by adding to $\mathcal{I}$ a variant of the d_2 rule:

$$\frac{\begin{array}{rcl} p_1[p_2] & \leq & p_3[p_4] \\ p_2[\overline{p_1}^{\,a}] & \leq & p_4[\overline{p_3}^{\,a'}] \end{array}}{p_1/p_2 \leq p_3/p_4}\ [d_{2b}] \qquad \begin{array}{l} p_2 \equiv \mathrm{a}::N/p_2' \\ p_4 \equiv \mathrm{a'}::N/p_4' \end{array}$$

Intuitively, the idea is to push left and right contexts into qualifiers, in order to propagate the information toward the inference process. The notation $\overline{p}^{\,\alpha}$ correspond to the application of a small inductive transformation function that computes a reversed xpath expression. It has the following property for all paths p

$$p/\mathrm{a}::N/\overline{p}^{\,a} \leq .$$

It is formally (and explicitly) defined by the function of figure 1, and it is based on the fact that each a axis has its own reverse axis [6] noted $\bar{a}$, such as shown below (see also [1] for another work using this symmetry property); Also the reader can take a closer look at the reversed **for** expression, the most complex one. Notice that it involves a fresh variable (a new one, not present in the reversed sub-term).

a		$\bar{a}$
self	$\leftrightarrow$	*self*
child	$\leftrightarrow$	*parent*
descendant	$\leftrightarrow$	*ancestor*
following	$\leftrightarrow$	*preceding*
following-sibling	$\leftrightarrow$	*preceding-sibling*
attribute	$\rightarrow$	*parent*
namespace	$\rightarrow$	*parent*

5.9 Capturing contradictions

Another category of difficult containment cases is illustrated by

$$\begin{array}{rcl} a[b/c][\mathbf{not}\ b] & \leq & \bot \\ a[\mathbf{not}\ b]/b & \leq & \bot \end{array}$$

where the qualifiers, being contradictory, force the left hand path to always evaluate to the empty set.

[6]Note also that due to the intrinsic asymmetry of *attribute* and *namespace* axes [6], we do not have $\bar{\bar{p}} = p$

We propose now two rules g_{2a} and g_{2b} that address these cases

$$\frac{\mathbf{true} \Rightarrow p \sqsubseteq \bot}{p \leq \bot}\ [g_{2a}] \qquad \frac{\mathbf{not}\ q_1 \Rightarrow q_2}{\mathbf{true} \Rightarrow p[q_1\ \mathbf{and}\ q_2] \sqsubseteq \bot}\ [g_{2b}]$$

One can prove the example above

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\mathbf{not}\ (b \sqsubseteq \bot) \Rightarrow \mathbf{not}\ (b \sqsubseteq \bot)}\ [t_2]}{\mathbf{true} \Rightarrow a[(b \sqsubseteq \bot)\ \mathbf{and}\ \mathbf{not}\ (b \sqsubseteq \bot)] \sqsubseteq \bot}\ [g_{2b}]}{\mathbf{true} \Rightarrow a[b \sqsubseteq \bot][\mathbf{not}\ (b \sqsubseteq \bot)] \sqsubseteq \bot}}{\mathbf{true} \Rightarrow a[b \sqsubseteq \bot][b] \sqsubseteq \bot}}{\mathbf{true} \Rightarrow a[b \sqsubseteq \bot]/b \sqsubseteq \bot}}{a[b \sqsubseteq \bot]/b \leq \bot}\ [g_{2a}]}{a[\mathbf{not}\ b]/b \leq \bot}$$

Note that we used a theorem t_2, easy to prove using e_i rules

$$\frac{}{q \Rightarrow q}\ [t_2]$$

5.10 Comparing Relative and Absolute paths

So far this paper did not address mixed containments, e.g. $a/b \leq \wedge//b$ which should be obviously established in this particular case. Note that for the XPath language, a relative path is not just a path that do not start with the root. Indeed, $a/ \wedge /b$ is equivalent to $\wedge[desc::a]/b$, and is thus an absolute path. Thus a relative path must not contain any $\wedge$ node, excepted in qualifiers, and we note $\tilde{p}$ such paths. The following rule, as a first attempt, seems appropriate to address mixed containment

$$\frac{N_1 \leq N_2}{\tilde{p}/\mathrm{a}::N_1 \leq \wedge/descendant::N_2}\ [g_3]$$

A rule like g_3 indeed allows proving $a/b \leq \wedge//b$

$$\cfrac{\cfrac{\cfrac{b \leq b}{a/b \leq \wedge/desc::b}\ [g_3]}{a/b \leq \wedge/desc::b \mid\ \wedge/self::b}\ [c_{2a}]}{a/b \leq \wedge/(desc::b|self::b)}}{a/b \leq \wedge//b}$$

However, it cannot be used for proving more subtle cases like

$$\begin{array}{lcl} parent::a/parent::b & \leq & \wedge/desc::b[a] \\ a/b & \leq & \wedge[.//a]/desc::* \end{array}$$

So we propose a more powerful variant that covers these cases

$$\frac{N_1 \leq N_2 \quad .[\wedge[.//\tilde{p}/a::N_1]]\ [\tilde{\bar{p}}^{\,a}] \leq .[\wedge[q_1]]\ [q_2]}{\tilde{p}/a::N_1 \leq \wedge[q_1]/desc::N_2[q_2]}\ [g_{3b}]$$

5.11 Synthesis

Considering increasingly complex cases, we tried to guide the reader through the difficulties of xpath containment, and also through the difficulty of building a powerful yet simple inference system. Still at this point the system is far from being complete. The (formal) soundness is being checked, and partially achieved. The next stage will be to achieve a (more difficult) completeness proof, or to understand were the remaining difficulties lie, because we actually don't know if the whole system, comprising rewriting extension proposed next section, is complete.

Before reaching this stage, some simple cases are simply not covered. We would like to show for instance that

$$a/.. \leq .$$

or that

$$following\text{-}sibling::*/preceding\text{-}sibling::a \leq ../a$$

We believe that such cases should not be addressed through extending the inference system. Indeed, this last must be kept as small as possible, and thus, mathematically tractable. Another possibility could be to enrich the equivalence relation. But in order to be useful and clear, the equivalence must stay quite intuitive and should not involve too complex and too much articulated transformations. Such operations must be clearly decomposed and even justified in order to be used for inferences.

We propose to address remaining containment cases through transforming the xpath expression into a normal form, more suited to handle comparisons. Such transformation can be achieved thank to a rewriting process, often mathematically more tractable and easier to understand than algorithms. We describe now the transformation architecture which allows us to go one step forward.

6 Transformation Architecture

6.1 overview

We propose a three stage transformation architecture that preserves the semantics of xpath expressions. The first one is a rewriting that bring any xpath under a disjunctive normal form $p_1|\cdots|p_n$ by applying as much as possible rules from a set $\mathcal{N}$, just derived from the equivalence $\equiv$. This stage strongly reduces the syntactic complexity, so that less rules have to be defined in subsequent stages. When applying as much as possible rules of $\mathcal{N}$, this gives a rewriting closure (or relation) noted $\mathcal{N}^*$.

The second transformation is a mixing of three rewriting:

1. $\mathcal{A}^*$ performs axis rewriting in order to simplify and eliminate steps as much as possible.

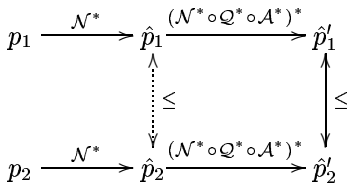
2. $\mathcal{Q}^*$ performs qualifier simplification and elimination

3. $\mathcal{N}^*$, as defined above, normalizes expressions reintroduced by the right hand sides of $\mathcal{A}$ and $\mathcal{Q}$ rules.

The combination of the three rewriting stages is noted $(\mathcal{N}^* \circ \mathcal{Q}^* \circ \mathcal{A}^*)^*$, where $\circ$ is the standard functional composition. This scheme expresses that we apply first all rules from $\mathcal{A}$ as much as possible, then the same for $\mathcal{Q}$ then the same for $\mathcal{N}$; the whole process itself is iterated as much as possible, before reaching a terminal form $\hat{p}$

The whole picture is captured by the following di-

agram



6.2 Disjunctive normal form

The rewriting relation $\mathcal{N}^*$, reduces any term p to a normal form $\hat{p}$, that is semantically equivalent, but syntactically simplified. the rule set $\mathcal{N}$ is the union of several subsets

$$\mathcal{N} = \mathcal{N}_1 \cup \cdots \mathcal{N}_6$$

Rules of $\mathcal{N}_1$ (fig. 8) are used in order to bring xpath expressions under a disjunctive form, i.e. where the disjunction symbol is always on top of the term, while qualifiers are brought under conjunctive form through rules of $\mathcal{N}_2$ and $\mathcal{N}_3$ (see fig. 9 and 10). Moreover, many expressions equivalent to $\bot$ can be detected at this early stage and propagated to top level thank to subset $\mathcal{N}_4$ (see fig. 11). Subsets $\mathcal{N}_5$ and $\mathcal{N}_6$ (see figures 4, 5) are dedicated to syntactic sugar expansion.

The rewriting can be applied for instance to $a/(b|c)[d]$:

$$\begin{array}{rl} a/(b|c)[d] & \xrightarrow{r_{1c}} \quad a/(b[d] \mid c[d]) \\ & \xrightarrow{r_{1b}} \quad a/b[d] \mid a/c[d] \end{array}$$

A subset of $\mathcal{N}$ is dedicated to the expansion of syntactic sugars, so that for instance,

$$\begin{array}{rl} & a/b[d] \\ \xrightarrow{r_{5d}, r_{5d}, r_{5d}} & child::a/child::b[child::d] \\ \xrightarrow{r_{6a}} & child::a/child::b[\mathbf{not}\ (child::d \sqsubseteq \bot)] \end{array}$$

Moreover, qualifiers are inserted so that each step is finally associated with at least one "true" qualifier

$$\begin{array}{rl} & child::a/child::b[\mathbf{not}\ (child::d \sqsubseteq \bot)] \\ \xrightarrow{*} & child::a[\mathbf{true}]/child::b[\mathbf{not}\ (child::d[\mathbf{true}] \sqsubseteq \bot)] \end{array}$$

6.3 Rewriting Axis and Qualifiers: $\mathcal{A}^*$ and $\mathcal{Q}^*$

The idea here is to apply rules that suppress steps as much as possible, while preserving the semantics equivalence. This operation is to be applied both to the principal axes and to axes embedded in qualifiers. For instance, $a/..$ can be transformed into $.[a]$, which opens now the possibility to use $\mathcal{I}$ in order to prove that $.[a] \leq .$ This is captured by the rules

$$\frac{p_1 \xrightarrow{\mathcal{T}^*} p'_1 \quad p'_1 \leq p_2}{p_1 \leq p_2}\ [h_1] \qquad \frac{p_2 \xrightarrow{\mathcal{T}^*} p'_2 \quad p_1 \leq p'_2}{p_1 \leq p_2}\ [h_2]$$

As an application example, we have

$$\dfrac{A \quad \dfrac{\dfrac{}{self::* \leq self::*}\,[c_2] \qquad \dfrac{}{child::a \not\sqsubseteq \bot \Rightarrow \mathbf{true}}\,[e_{3a}]}{self::*[child::a \not\sqsubseteq \bot] \leq self::*[\mathbf{true}]}\,[d_3]}{self::*[child::a \not\sqsubseteq \bot] \leq .}$$
$$a/.. \leq .\ [h_1]$$

with

$$A \equiv a/.. \xrightarrow{\alpha} self::*[\mathbf{not}\ (child::a \sqsubseteq \bot)]$$

In this expression, α represent the concatenation of all rule names (in the application order) that have been applied to derive the normal form. This derivation can thus be checked in order to verify this assertion.

Simplifying principal axis. In the example above, we used the rule

$$(r_{41}) \qquad \begin{array}{l} a \in \{child, attribute, namespace\} \\ \left\{ \begin{array}{ll} & a::N_1[q_1]/parent::N_2[q_2] \\ \rightarrow & self::N_2[q_2][a::N_1[q_1]] \end{array} \right. \end{array}$$

Actually, many rules are required in order to achieve complete axis simplification. A methodical exploration requires considering all possible pairs of axis, ignoring the redundant ones, i.e. those which can be expressed through other axes.

To this purpose, the first rule set $\mathcal{A}_1$ is dedicated to axis name suppression, as shown by figure 12, and

applies to *descendant-or-self, ancestor-or-self, following* and *preceding* axes.

The second rule set $\mathcal{A}_2$ (Fig. 13) derives empty expressions induced by axis composition, such as $attribute :: N/child :: N$ which always evaluates to empty node set since an attribute has no children.

Then come nine rule sets, $\mathcal{A}_3$ up to $\mathcal{A}_{12}$, dedicated to each of the nine remaining axes. Rules are proposed only when it is possible to gain structural simplifications. For instance,

$$child::a[q_1]/child::b[q_2]$$

clearly cannot be simplified in this sense, whereas

$$desc::a/child::b \rightarrow desc::b[parent::a]$$

brings a structural simplification, by pushing information into qualifiers. Some rules require using the powerful **for** construct, which captures current context and enables useful rewriting, as already shown in [1, section 5]. Let us consider the following (wrong) rewriting

$$child::a/desc::b \rightarrow desc::b[ancestor::a]$$

which is not sound (more precisely, it doesn't preserve the exact semantics, since the new expression selects more nodes that the old one). Indeed the exact rewriting has to use the for construct:

$child::a/desc::b \quad \rightarrow$
for $\$v$ **in** $child::a$ **return** $desc::b[(. \sqsubseteq \$v/desc::b)]$

The rules of $\mathcal{A}$ must be applied with a left-to-right implicit strategy in order to be sound, because this is the natural orientation of the / operation (which is of course not commutative).

Qualifier simplification. The purpose of the $\mathcal{A}^*$ rewriting is to push information into qualifiers in order to simplify axes. Now we propose to simplify qualifiers through another rewriting, noted $\mathcal{Q}^*$. Let us consider the following case as an illustration of this process

$$child::b[self::*[c]/d \not\sqsubseteq \bot)] \rightarrow child::b[c \textbf{ and } d]$$

Note that the complexity of the principal axis is unchanged, whereas the complexity of the inner path (embedded in the qualifier) is decreased.

7 Conclusion and perspectives

In this paper, we proposed a new approach for the study of the containment problem based on both a logical and rewriting approach. We have first introduced an inference system $\mathcal{I}^{\equiv}$ which allows asserting and proving properties on XPath expressions with the help of an equivalence relation $\equiv$. In order to keep this inference system reasonably small, we have proposed a multi-stage re-writing architecture. The first stage produces a disjunctive normal form, on which it is simpler to define the other stages, roughly, simplification rules that bring terms p under a form more suitable for comparisons, $\hat{p}$. Notably, it should be possible to infer containment on such terms without the help of the equivalence relation. This may help tackling problems introduced for instance by commutative laws such as q_1 **and** $q_2 \equiv q_2$ **and** q_1, which potentially involve infinite proof tree developments.

We intentionally tried to address the XPath language itself rather than a model or a too restrictive subset, even if it is a challenging task. This allows the study of properties on realistic path expressions and also to check the feasibility of rewriting based applications.

We are currently investigating mathematical proofs for properties such as soundness, completeness of the inference system $\mathcal{I}$ (which doesn't use equivalence but works on reduced terms $\hat{p}$), and also termination and soundness of the the rewriting architecture. We formulated the soundness as $(\forall\, \hat{p_1}, \hat{p_2}, x, \gamma)$

Proposition 1 *Soundness*

$$\gamma \vdash \hat{p_1} \leq \hat{p_2} \quad \wedge \quad \gamma \Vdash \phi \quad \Rightarrow \quad \mathcal{S}[\![\hat{p_1}]\!]^{\phi}_x \subseteq \mathcal{S}[\![\hat{p_2}]\!]^{\phi}_x$$

The expression $\gamma \Vdash \phi$ means that ϕ and γ are in conformance with respect to the definitions they contain. We are exploring a proof using a double structural induction based on the following induction hypothesis ($|p|$ is a suitable structural complexity measure)

$\mathcal{H}_i$

$$\begin{cases} \gamma \vdash p_1 \leq p_2 \quad \wedge \quad \gamma \Vdash \phi \quad \Rightarrow \quad \mathcal{S}[\![p_1]\!]^{\phi}_x \subseteq \mathcal{S}[\![p_2]\!]^{\phi}_x \\ \gamma \vdash q_1 \Rightarrow q_2 \quad \wedge \quad \gamma \Vdash \phi \quad \Rightarrow \quad \mathcal{Q}[\![q_1]\!]^{\phi}_x \Rightarrow \mathcal{S}[\![q_2]\!]^{\phi}_x \\ |p_1| + |q_1| \leq i \end{cases}$$

The completeness theorem is certainly harder, and we actually don't know yet if it holds, however, we

plan to use a similar approach in order to explore the proof.

Proposition 2 *Completeness*

$$\mathcal{S}[\![\hat{p_1}]\!]_x^\phi \subseteq \mathcal{S}[\![\hat{p_2}]\!]_x^\phi \quad \wedge \quad \gamma \Vdash \phi \quad \Rightarrow \quad \gamma \vdash \hat{p_1} \leq \hat{p_2}$$

Recent results on containment undecidability give hints that completeness could not be reached. On the other hand, it is hard to evaluate the exact consequence of using explicit node set constraints in qualifiers, which constitutes an original approach. Anyway, we expect to learn more when addressing the completeness proof: to identify more precisely (*i*) the combination of XPath features responsible for higher computational complexity and (*ii*) the sources of incompleteness or undecidability.

Even though high computational complexity means in practice low performances, this probably may not be a problem for most of the containment applications such as static analysis, because most of the xpath expressions used in the real world seem to be simple and small.

In parallel to this formal work, we started the development of algorithms embedded in an experimental demonstrator, written in the python language [9]. This prototype implements the inference together with the re-writing system presented in this paper. We plan to release the demonstrator as soon as the formal properties are established and their limits clearly understood.

Last but not least, we believe the rewriting architecture could be completed with an XPath optimization stage. This one can use rules dedicated to redundancies elimination (e.g. $a[b$ **and** $b/c] \rightarrow a[b/c]$) or qualifier decomposition (e.g. $a[b/c] \rightarrow a[b[c]]$, computationally more efficient). Moreover, the normalization stage could use a real execution context including schema information in order to exploit additional equivalences. For instance, schema constraints may permit to rewrite a/b into $a/*$, while this does not hold in the general case.

Beside pure optimization, containment under schema validity assumption can also be addressed using $\mathcal{I}$ unchanged, provided that the normalization stage is extended in order to exploit schema information.

References

[1] Dan Olteanu and Holger Meuss and Tim Furche and Francois Bry, *"Symmetry in XPath"*, technical report, October 2001, Computer Science Institute, Munich, Germany.

[2] Alin Deutsch and Val Tannen, *"Containment of Regular Path Expressions under Integrity Constraints"*, Knowledge Representation Meets Databases, 2001.

[3] Frank Neven and Thomas Schwentick, *"XPath containment in the presence of disjunction, DTDs, and variables"*, International Conference on Database Theory, 2003.

[4] John E. Hopcroft and Rajeev Motwani and Jeffrey D. Ullman, *"Introduction to Automata Theory, Languages, and Computation"*, Addison Wesley, Second Edition, 2000.

[5] Gerome Miklau and Dan Suciu, *"Containment and Equivalence for an XPath Fragment (Extended Abstract)"*, Symposium on Principles of Databases Systems, 2002.

[6] XML Path Language (XPath) 2.0, W3C working draft, 15 November, 2002.

[7] Phil Wadler, *"A formal semantics of patterns in XSLT"*, Markup Technologies, 1999.

[8] Phil Wadler", *"Two semantics for XPath"*, http://www.cs.bell-labs.com/who/wadler/ topics/xml.html, 1999.

[9] Guido van Rossum, *The Python language home page* http://www.python.org .

Biography of Jean-Yves Vion-Dury Jean-Yves holds an engineering degree from Conservatoire National des Arts et Métiers (1993) and a Ph.D. in Computer Science from Université Joseph Fourier (1999).

As an invited researcher, he recently joined the WAM project at INRIA, in charge of the scientific development of Ω, a new transformation language strongly based on XPath.

His previous research focused on the theoretical and practical design of Circus-DTE, a specialized programming language created around the paradigm of typed structure transformation, applied to XML document processing.

Jean-Yves joined the Xerox Research Center Europe (XRCE, Grenoble) in April 1995, as Ph.D. intern working in partnership with the french national research institute INRIA. He worked first on the Olan project, designing tools, environment and infrastructure for distributed component programming (language semantics, runtime architecture, compiler, debugger) and then on the CLF architecture (CLF stands for Coordination Language facilities). In April 1999, Jean-Yves joined the newly created DMTT group to apply the result of his former research on document transformation. Before joining Xerox research and Technologies, he worked for years in the industrial world, focusing on electrical engineering and applied computer sciences.

Biography of Nabil Layaïda Dr. Nabil Layaïda received a Ph.D. degree in computer science from university of Grenoble in 1997. Since 1998, he is working as a research officer at INRIA (Institut National de Recherche en Informatique et en Automatique). He is managing the presentation services for multimedia systems group within the WAM project. His research interests are in the fields of interactive structured and multimedia document processing, adaptive documents, multimedia applications for the mobile devices, structure transformations, temporal scheduling and synchronization and their applications on the Web. He is also a member of the SYMM working group and co-author of SMIL 1.0 and SMIL 2.0 recommendations (Synchronized Multimedia Integration Language) of the World Wide Web Consortium.

$$
\begin{array}{lcl}
\overline{p_1|p_2}^{\,\alpha} & = & \overline{p_1}^{\,\alpha}|\overline{p_2}^{\,\alpha} \\
\overline{p/a::N[q]}^{\,\alpha} & = & \bar{\alpha}::N[q]/\overline{p}^{\,a} \\
\overline{a::N}^{\,\alpha} & = & \bar{\alpha}::N/\bar{a}::* \\
\overline{\$v}^{\,\alpha} & = & \$v \\
\overline{\wedge}^{\,\alpha} & = & \wedge \\
\overline{\textbf{for } \$v \textbf{ in } p_1 \textbf{ return } p_2}^{\,\alpha} & = & \\
\multicolumn{3}{l}{\textbf{for } \$v \textbf{ in } ., \$w \textbf{ in } \overline{p_2}^{\,\alpha} \textbf{ return } \$w/\overline{p_1}^{\,\alpha} \qquad (\$w \notin p_1)}
\end{array}
$$

Figure 1: The reversed xpath computation

$$
\begin{array}{rcl}
\textit{child} & \leq & \textit{descendant} \\
\textit{parent} & \leq & \textit{ancestor} \\
\textit{following-sibling} & \leq & \textit{following} \\
\textit{preceding-sibling} & \leq & \textit{preceding}
\end{array}
$$

Figure 2: Partial ordering of axes

$$
\begin{array}{rcccl}
n & \leq & * & \leq & \textbf{node}() \\
\textbf{processing-instruction}(), \textbf{text}() & & & \leq & \textbf{node}() \\
\textbf{element}()(), \textbf{comment}() & & & \leq & \textbf{node}()
\end{array}
$$

Figure 3: Partial ordering of node tests

.	$\rightarrow$	$self::\mathbf{node}()$	(r_{5a})
..	$\rightarrow$	$parent::\mathbf{node}()$	(r_{5b})
$p_1//p_2$	$\rightarrow$	$p_1/desc::\mathbf{node}()/p_2 \mid p_1/p_2$	(r_{5c})
N	$\rightarrow$	$child::N$	(r_{5d})
$@N$	$\rightarrow$	$attribute::N$	(r_{5e})
$ns:n$	$\rightarrow$	$n[namespace::ns]$	(r_{5f})
if q **then** p_1 **else** p_2	$\rightarrow$	$p_1[q] \mid p_2[\mathbf{not}\ q]$	(r_{5g})

Figure 4: Syntactic sugars (axis) ($\mathcal{N}_5$)

p	$\rightarrow$	$\mathbf{not}\ (p \sqsubseteq \bot)$	(r_{6a})
$p_1 == p_2$	$\rightarrow$	$(p_1 \sqsubseteq p_2)\ \mathbf{and}\ (p_2 \sqsubseteq p_1)$	(r_{6b})
$p_1 \sqsubset p_2$	$\rightarrow$	$(p_1 \sqsubseteq p_2)\ \mathbf{and\ not}\ (p_2 \sqsubseteq p_1)$	(r_{6c})
$p_1 \sqsupset p_2$	$\rightarrow$	$p_2 \sqsubset p_1$	(r_{6d})
$p_1 \sqsupseteq p_2$	$\rightarrow$	$p_2 \sqsubseteq p_1$	(r_{6e})
$p_1 \neq p_2$	$\rightarrow$	$\mathbf{not}\ (p_1 == p_2)$	(r_{6f})

Figure 5: Syntactic sugars (qualifiers) ($\mathcal{N}_6$)

p	$\equiv$	$p[\mathbf{true}]$
p	$\equiv$	$./p$
$\bot$	$\equiv$	$p[\mathbf{false}]$
$p[q_1][q_2]$	$\equiv$	$p[q_1\ \mathbf{and}\ q_2]$
$p[q_1] \mid p[q_2]$	$\equiv$	$p[q_1\ \mathbf{or}\ q_2]$
$(p_1\|p_2)/p$	$\equiv$	$p_1/p \mid p_2/p$
$p/(p_1\|p_2)$	$\equiv$	$p/p_1 \mid p/p_2$

Figure 6: XPath equivalences

$\mathbf{not}\ (q_1\ \mathbf{and}\ q_2)$	$\equiv$	$\mathbf{not}\ q_1\ \mathbf{or\ not}\ q_2$
$\mathbf{not}\ (q_1\ \mathbf{or}\ q_2)$	$\equiv$	$\mathbf{not}\ q_1\ \mathbf{and\ not}\ q_2$
$q\ \mathbf{and}\ (q_1\ \mathbf{or}\ q_2)$	$\equiv$	$(q\ \mathbf{and}\ q_1)\ \mathbf{or}\ (q\ \mathbf{and}\ q_2)$
$q\ \mathbf{or}\ (q_1\ \mathbf{and}\ q_2)$	$\equiv$	$(q\ \mathbf{or}\ q_1)\ \mathbf{and}\ (q\ \mathbf{or}\ q_2)$
$\mathbf{not\ not}\ q$	$\equiv$	q
$\mathbf{not\ true}$	$\equiv$	$\mathbf{false}$
$\mathbf{not\ false}$	$\equiv$	$\mathbf{true}$
$\mathbf{true\ or}\ p$	$\equiv$	$\mathbf{true}$
$\mathbf{true\ and}\ p$	$\equiv$	p
$\mathbf{false\ and}\ p$	$\equiv$	$\mathbf{false}$
$\mathbf{false\ or}\ p$	$\equiv$	p
$q_1\ \mathbf{and}\ q_2$	$\equiv$	$q_2\ \mathbf{and}\ q_1$
$q_1\ \mathbf{or}\ q_2$	$\equiv$	$q_2\ \mathbf{or}\ q_1$

Figure 7: Qualifier equivalences

$(p_1\|p_2)/p$	$\rightarrow$	$p_1/p \mid p_2/p$	(r_{1a})
$p/(p_1\|p_2)$	$\rightarrow$	$p/p_1 \mid p/p_2$	(r_{1b})
$(p_1\|p_2)[q]$	$\rightarrow$	$p_1[q] \mid p_2[q]$	(r_{1c})
$p[q_1\ \mathbf{or}\ q_2]$	$\rightarrow$	$p[q_1] \mid p[q_2]$	(r_{1d})
$\mathbf{for}\ \$v\ \mathbf{in}\ p_1\|p_2\ \mathbf{return}\ p$	$\rightarrow$		
$\quad \mathbf{for}\ \$v\ \mathbf{in}\ p_1\ \mathbf{return}\ p$			
$\quad \mid \mathbf{for}\ \$v\ \mathbf{in}\ p_2\ \mathbf{return}\ p$			(r_{1e})
$\mathbf{for}\ \$v\ \mathbf{in}\ p\ \mathbf{return}\ p_1\|p_2$	$\rightarrow$		
$\quad \mathbf{for}\ \$v\ \mathbf{in}\ p\ \mathbf{return}\ p_1$			
$\quad \mid \mathbf{for}\ \$v\ \mathbf{in}\ p\ \mathbf{return}\ p_2$‘			(r_{1f})
$(\mathbf{for}\ \$v\ \mathbf{in}\ p_1\ \mathbf{return}\ p_2)/p$	$\rightarrow$		
$\quad \mathbf{for}\ \$v\ \mathbf{in}\ p_1\ \mathbf{return}\ (p_2/p)$			(r_{1g})
$p/(\mathbf{for}\ \$v\ \mathbf{in}\ p_1\ \mathbf{return}\ p_2)$	$\rightarrow$		
$\quad \mathbf{for}\ \$v\ \mathbf{in}\ p/p_1\ \mathbf{return}\ (p/p_2)$			(r_{1h})

Figure 8: $\mathcal{N}_1$: distributing | and /

$$
\begin{array}{lcll}
(p_1/p_2)[q] & \to & p_1/(p_2[q]) & (r_{2a}) \\
p[q_1 \textbf{ or } q_2] & \to & p[q_1] \mid p[q_2] & (r_{2b}) \\
p[q_1][q_2] & \to & p[q_1 \textbf{ and } q_2] & (r_{2c}) \\
(\textbf{for } \$v \textbf{ in } p_1 \textbf{ return } p_2)[q] & \to & & \\
\textbf{for } \$v \textbf{ in } p_1 \textbf{ return } (p_2[q]) & & & (r_{2d}) \\
\mathrm{a}::N/p & \to & \mathrm{a}::N[\textbf{true}]/p & (r_{2e}) \\
\wedge/p & \to & \wedge[\textbf{true}]/p & (r_{2f}) \\
p/\wedge[q_1] & \to & \wedge[q_1][.//p] & (r_{2g}) \\
\$v/p & \to & \$v[\textbf{true}]/p & (r_{2h}) \\
p/\$v[q] & \to & & \\
\textbf{for } \$w \textbf{ in } p \textbf{ return } \$v[q][\$w] & & & (r_{2i})
\end{array}
$$

Figure 9: Nesting qualifiers with $\mathcal{N}_2$

$$
\begin{array}{lcll}
\textbf{not not } q & \to & q & (r_{3a}) \\
\textbf{not } (q_1 \textbf{ and } q_2) & \to & (\textbf{not } q_1) \textbf{ or } (\textbf{not } q_2) & (r_{3b}) \\
\textbf{not } (q_1 \textbf{ or } q_2) & \to & (\textbf{not } q_1) \textbf{ and } (\textbf{not } q_2) & (r_{3c}) \\
q \textbf{ and } (q_1 \textbf{ or } q_2) & \to & (q \wedge q_1) \textbf{ or } (q \wedge q_2) & (r_{3d}) \\
q \textbf{ or } (q_1 \textbf{ and } q_2) & \to & (q \textbf{ or } q_1) \textbf{ and } (q \textbf{ or } q_2) & (r_{3e}) \\
(q_1 \textbf{ or } q_2) \textbf{ and } q & \to & (q_1 \wedge q) \textbf{ or } (q_2 \wedge q) & (r_{3f}) \\
(q_1 \textbf{ and } q_2) \textbf{ or } q & \to & (q_1 \textbf{ or } q) \textbf{ and } (q_2 \textbf{ or } q) & (r_{3g}) \\
\textbf{true and } q & \to & q & (r_{3h}) \\
\textbf{true or } q & \to & \textbf{true} & (r_{3i}) \\
\textbf{false and } q & \to & \textbf{false} & (r_{3j}) \\
\textbf{false or } q & \to & q & (r_{3k}) \\
 & & & \\
q \textbf{ and true} & \to & q & (r_{3l}) \\
q \textbf{ or true} & \to & \textbf{true} & (r_{3m}) \\
q \textbf{ and false} & \to & \textbf{false} & (r_{3n}) \\
q \textbf{ or false} & \to & q & (r_{3o}) \\
 & & & \\
\textbf{not true} & \to & \textbf{false} & (r_{3p}) \\
\textbf{not false} & \to & \textbf{true} & (r_{3q}) \\
 & & & \\
\text{if } p \neq p'|p'' & & & \\
p \sqsubseteq (p_1|p_2) & \to & (p \sqsubseteq p_1) \textbf{ or } (p \sqsubseteq p_2) & (r_{3r}) \\
 & & & \\
(p_1|p_2) \sqsubseteq p & \to & (p_1 \sqsubseteq p) \textbf{ and } (p_2 \sqsubseteq p) & (r_{3s}) \\
\bot \sqsubseteq p & \to & \textbf{true} & (r_{3t}) \\
p \sqsubseteq p & \to & \textbf{true} & (r_{3u})
\end{array}
$$

Figure 10: Conjunctive form for qualifiers ($\mathcal{N}_3$)

$$
\begin{array}{lcll}
\bot/p & \to & \bot & (r_{4a}) \\
p/\bot & \to & \bot & (r_{4b}) \\
\bot|p & \to & p & (r_{4c}) \\
p|\bot & \to & p & (r_{4d}) \\
\bot[q] & \to & \bot & (r_{4e}) \\
p[\textbf{false}] & \to & \bot & (r_{4f}) \\
\textbf{for } \$v \textbf{ in } p \textbf{ return } \bot & \to & \bot & (r_{4g}) \\
\textbf{for } \$v \textbf{ in } \bot \textbf{ return } p & \to & \bot & (r_{4h}) \\
 & & & \\
a \neq \mathit{child},\mathit{descendant} & & & \\
\wedge[\cdots(\mathrm{a}::N[q]/p \not\sqsubseteq \bot)\cdots] & \to & \bot & (r_{4i}) \\
 & & & \\
n \neq n' & & & \\
\mathrm{a}::n[\cdots(\mathit{self}::n'[q]/p \not\sqsubseteq \bot)\cdots] & \to & \bot & (r_{4j})
\end{array}
$$

Figure 11: Detecting void paths ($\mathcal{N}_4$)

$$
\begin{array}{rcll}
\mathit{desc\text{-}or\text{-}self}::N & \to & \mathit{desc}::N|\mathit{self}::N & r_{a11} \\
\mathit{anc\text{-}or\text{-}self}::N & \to & \mathit{anc}::N|\mathit{self}::N & r_{a12} \\
\mathit{following}::N & \to & \mathit{anc\text{-}or\text{-}self}::* & r_{a13} \\
 & & /\mathit{following\text{-}sibling}::* & \\
 & & /\mathit{desc\text{-}or\text{-}self}::N & \\
\mathit{preceding}::N & \to & \mathit{anc\text{-}or\text{-}self}::* & r_{a14} \\
 & & /\mathit{preceding\text{-}sibling}::* & \\
 & & /\mathit{desc\text{-}or\text{-}self}::N &
\end{array}
$$

Figure 12: Suppression of axes ($\mathcal{A}_1$)

$$
\begin{array}{rcll}
\wedge[q]/\mathrm{a}::N[q'] & \to & \bot & r_{a21} \\
 & & & \\
a \neq \mathit{self},\mathit{parent},\mathit{ancestor} & & & \\
\mathit{attribute}::N_1[q_1]/\mathrm{a}::N_2[q_2] & \to & \bot & r_{a22} \\
\mathit{namespace}::N_1[q_1]/\mathrm{a}::N_2[q_2] & \to & \bot & r_{a23} \\
 & & & \\
n_2 \neq n_1 & & & \\
\mathrm{a}::n_1[q_1]/\mathit{self}::n_2[q_2] & \to & \bot & r_{a24} \\
 & & & \\
p_1/\mathit{self}::*[q] & \to & p_1[q] & r_{a25}
\end{array}
$$

Figure 13: Detection of empty paths ($\mathcal{A}_2$)

$p_1/self::*[q]$ $\rightarrow$ $p_1[q]$
$\wedge[q_1]/\wedge[q_2]$ $\rightarrow$ $\wedge[q_1][q_2]$

$\exists N = \min(N_1, N_2)$
$a::N_1[q_1]/self::N_2[q_2]$ $\rightarrow$ $a::N[q_1][q_2]$

$self::*[\mathbf{true}]/p$ $\rightarrow$ p
$desc::*[\mathbf{true}]/desc::N[q]$ $\rightarrow$ $*/desc::N[q]$
$anc::*[\mathbf{true}]/anc::N[q]$ $\rightarrow$ $parent::*/anc::N[q]$

Figure 14: axis simplification ($\mathcal{A}_3$)

if a is	then $a::N_1[q_1]/a'::N_2[q_2]$ rewrites to	#
	if a' is *parent*	
child, *attribute*, *namespace*	$self::N_2[q_2][a::N_1[q_1]]$	r_{41}
descendant	$desc\text{-}or\text{-}self::N_2[q_2][N_1[q_1]]$	r_{42}
f-sibling, *p-sibling*	$.[a::N_1[q_1]]/parent::N_2[q_2]$	r_{43}
	if a' is *child*	
parent	$(p\text{-}sibling::N_2 \mid f\text{-}sibling::N_2 \mid self::N_2)[q_2][parent::N_1[q_1]]$	r_{44}
ancestor r_{a31}	**for** $\$v$ **in** . **return** $\wedge/desc::N_2[q_2][parent::N_1[q_1][Q]]$ with $Q \equiv (\$v \sqsubseteq desc\text{-}or\text{-}self::*)$	r_{45}
r_{a32}	if a' is *descendant*	
parent	$a::N_1[q_1]/child::N_2[q_2]$ $a::N_1[q_1]/*/desc::N_2[q_2]$	r_{46}
r_{a33} *ancestor* r_{a34}	**for** $\$v$ **in** . **return** $\wedge/desc::N_2[q_2][ancestor::N_1[q_1][Q]]$ with $Q \equiv (\$v \sqsubseteq desc\text{-}or\text{-}self::*)$	r_{47}
r_{a35}	if a' is *ancestor*	
r_{a36} *child*, *attribute* *namespace*	$.[a::N_1[q_1]]/anc\text{-}or\text{-}self::N_2[q_2]$	r_{48}
descendant	$anc\text{-}or\text{-}self::N_2[q_2] \mid desc::N_2[q_2][desc::N_1[q_1]]$	r_{49}
f-sibling, *p-sibling*	$.[a::N_1[q_1]]/ancestor::N_2[q_2]$	r_{4a}
	if a' is *f-sibling*,*p-sibling*	
child, *descendant*	$a::N_2[q_2][\bar{a}'::N_1[q_1]]$	r_{4b}
$\bar{a}'$	**for** $\$v$ **in** . **return** $P[q_2][Q_1]$ with $P \equiv (a::N_2 \mid self::N_2 \mid \bar{a}::N_2[Q_2])$ $Q_1 \equiv a::N_1[q_1]$ $Q_2 \equiv (\$v \sqsubseteq \bar{a}::*)$	r_{4c}

Figure 15: Elimination of opposed axis ($\mathcal{A}_4$)

$$
\begin{array}{lll}
 & a :: N_1[\cdots(\bar{a} :: N_2[q]/p \not\sqsubseteq \bot)\cdots] & \\
\rightarrow & \mathit{self} :: N_2[q][p]/a :: N_1[\cdots] & (r_{q1}) \\
 & & \\
 & \exists N = \min(N_1, N_2) & \\
 & a :: N_1[\cdots(\mathit{self} :: N_2[q]/p \not\sqsubseteq \bot)\cdots] & \\
\rightarrow & a :: N[\cdots][q][p][\cdots] & (r_{q2})
\end{array}
$$

Figure 16: Qualifier simplification ($\mathcal{Q}$)