XPath on Left and Right Sides of Rules: Toward Compact XML Tree Rewriting through Node Patterns

Jean-Yves Vion-Dury[*]

WAM project[17]

INRIA Rhône-Alpes
655 Avenue de l'Europe
Montbonnot, France

Jean-Yves.Vion-Dury@{inria.fr,xrce.xerox.com}

ABSTRACT

XPath [3, 5] is a powerful and quite successful language able to perform complex node selection in trees through compact specifications. As such, it plays a growing role in many areas ranging from schema specifications, designation and transformation languages to XML query languages. Moreover, researchers have proposed elegant and tractable formal semantics [8, 9, 10, 14], fostering various works on mathematical properties and theoretical tools [10, 13, 12, 14]. We propose here a novel way to consider XPath, not only for selecting nodes, but also for tree rewriting using rules. In the rule semantics we explore, XPath expressions (noted p, p') are used both on the left and on the right side (i.e. rules have the form $p \rightarrow p'$). We believe that this proposal opens new perspectives toward building highly concise XML transformation languages on widely accepted basis.

Categories and Subject Descriptors

D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*document transformation, XPath, tree rewriting*

General Terms

Language Theory Design

1. INTRODUCTION

Rewriting is a deeply studied branch of theoretical computer science, and influenced an impressive amount of research work, ranging from the general computation theory up to practical applications. We assume in this paper that the reader owns some basic knowledge on rewriting theory and also on XPath (see [1, 3] for an exhaustive introduction) As a computation model, it has some good properties that attracted the attention of scientists looking for structure transformation languages : decomposition in small basic units (rules), structural matching/filtering, modularity, and static analysis through unification (see for instance termination proofs based on critical peaks analysis, itself based on rule's terms unification [1]).

Rewriting has been applied to string and tree transformation in compiler construction, and also to more complex data structures such as (labelled) graph transformation. However, it has been poorly applied to XML transformation, partly because the richness of the XML tree model require quite complex tree pattern matching specifications [1]. In particular, conventional tree patterns requires specifying not only the interesting sub-structures, but also the ones which have to be ignored during the matching.

XPath in contrast is a node selection language that brings a great simplicity by focusing on paths instead of structures, making powerful hypothesis and metaphors such as implicit context node and navigation. It is today a widely known specification, adopted by many programmers and also recognized as being mathematically tractable [8, 10, 13, 12, 14].

This paper is based on the idea of reusing XPath in systems of rules like $p \rightarrow p'$, where p and p' are XPath like expressions. In order to build such systems, one has to address two problems:

1. how should XPath be extended in order to embed the filtering variables required for memorizing and reusing sub-structures ?

2. how should an XPath expression be interpreted on the right hand side (p') with respect to tree construction and modification?

Wolfgang May [16] proposed to extend XPath toward a logic based approach, and to describe syntactic and semantic modifications that enable filtering through unification variables. The proposed approach somehow solves the two problems above, but requires a deep modification in the orig-

[*]Visiting researcher from Xerox Research Centre Europe.

[1]and also partly because many XML transformations require complex rule application schemes

inal syntax and semantics. Moreover, the latter is deeply related to logic and prolog-like execution model, which makes harder to understand the relationship with the pure XPath semantics and the structure of the computation. In addition, this language addresses only monotonic transformations of trees (nodes and links are never deleted along transformations).

Section 2 presents a formal definition of the syntax and semantics of XPath, inspired from [14] and slighly extended toward handling positional information and node value tests. Then we introduce the general notion of path containment, and give a mathematical definition of this property.

The notion of pattern and rule semantics is defined in section 3, and related to XPath semantics and containment through a translation function Ξ that transforms an XPath pattern $\vec{p}$ into an XPath expression p. Thanks to this approach, the pattern matching operation is nothing else than finding an environment in which p will select the context node itself. A small and simple set of operations for tree modification is proposed in order to define the semantics of rule application, i.e. the meaning of the right hand side.

We illustrate the expressiveness of the approach in section 4, with realistic examples from MathML [6].

We conclude through a short synthesis and some perspectives on our future work on the topic.

2. XPATH: A FORMAL DEFINITION AND SOME PROPERTIES

The purpose of this section is to describe formally a language derived from a fragment of XPath 2.0 . We believe that this fragment is expressive enough for most "real world" applications. However,if it deserves today our research purpose, it is planned to extend it step by step as long as we are able to maintain its mathematical properties.

2.1 Syntax

The syntax is defined in two stages: *(i)* the core language, and *(ii)* the syntactic sugar, which just rewrites into expressions of the core. The following definition is a slight extension of [14]

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

Note that according to XPath 2.0 specification, and to proposal [10], but not in accordance with the proposal of [8], 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 takes 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} in figure 1).

The important extension we propose with respect to qualifiers is the node set inclusion constraint $p_1 \sqsubseteq p_2$, which brings extra expressive power and interesting possibilities for containment inference (see [14]). The inclusion also brings additionnal facilities for discriminating embedded structures and for extending the standard node set equality test in an

$$\begin{array}{lll l} . & \rightarrow & \textit{self}{::}\,\mathbf{node}() & (r_{5a}) \\ p_1//p_2 & \rightarrow & p_1/\textit{desc-or-self}{::}*/p_2 & (r_{5c}) \\ N & \rightarrow & \textit{child}{::}\,N & (r_{5d}) \\ @N & \rightarrow & \textit{attribute}{::}\,N & (r_{5e}) \\ ns{:}n & \rightarrow & n[\textit{namespace}{::}ns] & (r_{5f}) \\ p[?v] & \rightarrow & p[. == \$v] & (r_{5h}) \\ p[i] & \rightarrow & p[\mathbf{position}() = i] & (r_{5i}) \\ p[-i] & \rightarrow & p[\mathbf{position}() = -i] & (r_{5i}) \end{array}$$

Figure 1: Syntactic sugars ($\mathcal{N}_5$)

elegant way. For instance

$$\begin{array}{lll} q & ::= & \textbf{true} \;\mid\; \textbf{false} \;\mid\; (q) \;\mid\; \textbf{not}\; q \;\mid \\ & & q \;\textbf{and}\; q \;\mid\; q \;\textbf{or}\; q \;\mid\; p \sqsubseteq p \end{array}$$

However, we propose an extension of qualifier syntax so that minimal positional information can be expressed (i is the lexical representation for strictly positive integers), and also information on linearized content (s is the lexical representation for strings).

$$q \;::=\; \mathbf{position}() = i \;\mid\; \mathbf{position}() = -i \;\mid\; p = s$$

So, expressions like $a[2]/b[@att = \text{"test"}]$[2] are legal. A (partial) reflexive ordering of N elements is defined in figure 2.

$$\begin{array}{rcl} n \;\le\; * & \le & \mathbf{node}() \\ \mathbf{processing\text{-}instruction}(), \mathbf{text}() & \le & \mathbf{node}() \\ \mathbf{element}(), \mathbf{comment}() & \le & \mathbf{node}() \end{array}$$

Figure 2: Partial ordering of node tests

The a symbol denotes *axes*, ranging over the whole set defined in the W3C specification

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

We define the syntactic sugar (see figure 1 for their translation into core expressions)

$$\begin{array}{lll} p & ::= & p//p \;\mid\; . \;\mid\; N \;\mid\; @N \;\mid\; ns{:}N \;\mid \\ & & p[?v] \;\mid\; p[i] \;\mid\; p[-i] \end{array}$$

We have to comment right now on the $p[?v]$ notation, which will be our mechanism to introduce filtering variables in XPath expressions (see coming section 3.1). More precisely, it rewrites into $p[. == \$v]$, were $\$v$ is a free variable. Following the semantics of the $==$ node set operator (see section 2.2), it means that the matching requires the existence of a substitution assigning a suitable node value to the variable. It must thus satisfy set inclusion constraints.

We propose syntactic sugars for qualifier (see figure 3):

$$\begin{array}{lll} q & ::= & p \;\mid\; p == p \;\mid\; p \neq p \;\mid \\ & & p \sqsubset p \;\mid\; p \sqsupset p \;\mid\; p \sqsupseteq p \end{array}$$

[2]equivalent to $a[\mathbf{position}() = 2]/b[\textit{attribute}{::}\textit{att} = \text{"test"}]$ after syntactic sugar expansion

$$\begin{array}{lcll}
p & \to & \textbf{not } (p \sqsubseteq \bot) & (r_{6a}) \\
p_1 == p_2 & \to & (p_1 \sqsubseteq p_2) \textbf{ and } (p_2 \sqsubseteq p_1) & (r_{6b}) \\
p_1 \sqsubset p_2 & \to & (p_1 \sqsubseteq p_2) \textbf{ and not } (p_2 \sqsubseteq p_1) & (r_{6c}) \\
p_1 \sqsupset p_2 & \to & p_2 \sqsubset p_1 & (r_{6d}) \\
p_1 \sqsupseteq p_2 & \to & p_2 \sqsubseteq p_1 & (r_{6e}) \\
p_1 \neq p_2 & \to & \textbf{not } (p_1 == p_2) & (r_{6f})
\end{array}$$

Figure 3: Syntactic sugars (qualifiers) ($\mathcal{N}_6$)

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

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

so that for instance

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

is syntactically understood as

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

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

2.2 Semantics

We reuse the denotational definition of [14], also inspired from [8, 10], extended with a node set parameter S and with an execution context ϕ that uniquely associates variable names to tree nodes.

2.2.1 *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 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 [3]. 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 opening tags occurrence in the linear document. The ordering relation $x_1 \ll x_2$ is **true** if the opening tag of x_1 appears before the opening tag of x_2, **false** otherwise.

2.2.2 *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 of the XPath language.

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

$$\begin{array}{lcl}
\mathcal{S}[\![\bot]\!]_x^{\phi,S} & = & \emptyset \\
\mathcal{S}[\![(p)]\!]_x^{\phi,S} & = & \mathcal{S}[\![p]\!]_x^{\phi,S} \\
\mathcal{S}[\![p_1|p_2]\!]_x^{\phi,S} & = & \mathcal{S}[\![p_1]\!]_x^{\phi,S} \cup \mathcal{S}[\![p_2]\!]_x^{\phi,S} \\
\mathcal{S}[\![p_1/p_2]\!]_x^{\phi,S} & = & \{x_2 \mid x_1 \in \mathcal{S}[\![p_1]\!]_x^{\phi,S} \wedge x_2 \in \mathcal{S}[\![p_2]\!]_{x_1}^{\phi,S}\} \\
\mathcal{S}[\![p[q]]\!]_x^{\phi,S} & = & \text{let } S_1 = \mathcal{S}[\![p]\!]_x^{\phi,S} \text{ in} \\
 & & \{x_1 \mid x_1 \in s_1 \wedge \mathcal{Q}[\![q]\!]_{x_1}^{\phi,S_1}\} \\
\mathcal{S}[\![\$v]\!]_x^{\phi,S} & = & \{\phi(v)\} \\
\mathcal{S}[\![\mathrm{a}{::}N]\!]_x^{\phi,S} & = & \{x_1 \mid f_a(x) \wedge \mathcal{T}_a(x_1, N)\}
\end{array}$$

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

$$\begin{array}{l}
\mathcal{S}[\![\textbf{for } \$v \textbf{ in } p_1 \textbf{ return } p_2]\!]_x^{\phi,S} \\
= \;\; \{x_2 \mid x_1 \in \mathcal{S}[\![p_1]\!]_x^{\phi,S} \wedge x_2 \in \mathcal{S}[\![p_2]\!]_x^{\phi,v=x_1}\}
\end{array}$$

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

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

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**

2.2.3 *Semantics of qualifiers.*

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

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

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

The former definitions correspond exactly to those of [14], up to the function signature. We now define the positional

[3]Every node x of the tree is reachable from the root ; mathematically, $\wedge \rightarrowtriangle^+ x$

and linearization based extensions

$$\begin{array}{lcl}
\mathcal{Q}[\![\mathbf{position}() = i]\!]_x^{\phi,S} & = & \text{let } S_1 = \{x_1 \in S | x_1 \ll x\} \\
 & & \text{in } (size(S_1) = i - 1) \\
\mathcal{Q}[\![\mathbf{position}() = -i]\!]_x^{\phi,S} & = & \text{let } S_1 = \{x_1 \in S | x_1 \ll x\} \\
 & & \text{in } (size(S_1) = size(S) - i) \\
\mathcal{Q}[\![p = s]\!]_x^{\phi,S} & = & \text{let } S_1 = \mathcal{S}[\![p]\!]_x^{\phi,S} \text{ in} \\
 & & (S_1 \neq \emptyset) \wedge (\textbf{value-of}(S_1) = s)
\end{array}$$

Note also that string value tests ($p = s$) require non empty path selection. The **value-of** function returns the concatenation of all text leaves descendant of each node in the selection, and moreover, in document order.

2.3 Containment of XPath expressions

The containment relation $p \leq p'$ expresses that for all context node x, the node set resulting from the evaluation of p is included in the node set selected by p'. More precisely, this property is defined by

$$p \leq p' \text{ iff } \quad \forall x \quad \mathcal{S}[\![p]\!]_x^{\emptyset,\emptyset} \subseteq \mathcal{S}[\![p']\!]_x^{\emptyset,\emptyset}$$

The containment inference (see [14]) requires actually a context γ that conveys information about variables (it may be considered as empty, and just ignored as above).

For instance (example taken from [14]), one can prove that $\$v/b \leq (a|c)/b$ in a context $\gamma = \{v\!:\!c|a\}$

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

In order to express the containment property in a quite general way, the static environment γ and the execution environment ϕ must be considered in conformance ($\gamma \Vdash \phi$)

Property 1. *Conformance of an execution environment*

$$\gamma \Vdash \phi \textit{ iff } \quad \forall v \in \gamma, \quad \exists x, S \quad \phi(v) \subseteq \mathcal{S}[\![\gamma(v)]\!]_x^{\phi,S}$$

Thus a containment assertion $\gamma \vdash p \leq p'$ is expressed through the following general property

Property 2. *Containment.*

$$\gamma \vdash p \leq p' \textit{ iff } \quad \forall x, \phi, S \quad \gamma \Vdash \phi \implies \mathcal{S}[\![p]\!]_x^{\phi,S} \subseteq \mathcal{S}[\![p']\!]_x^{\phi,S}$$

It turns out to be a quite exciting and fundamental problem, and also difficult from the computational point of view (see [13, 12] for model-based approaches, and our later work [14] for a syntactic-based approach). We propose to use this relation in order to define the semantics of rules, and also to establish later some usefull structural properties.

3. PATTERNS AND RULES

Rules have the form $\vec{p} \rightarrow \vec{p}'$, where $\vec{p}$ denotes an XPath pattern, i.e. an expression that contains free variables that must be instanciated in order the pattern to match. The matching function $\mathcal{M}$ returns a context ϕ' that defines a value to each free variable of the pattern.

A single rule is evaluated in a context including the so-called "contextual node" (which defines the notion of relative location in the tree, just as in the standard XPath semantics), and an execution environment ϕ that uniquely defines the value of bound variables.

The left hand side is evaluated in the current node context, and if a matching is found, the tree is transformed by executing a sequence of operations, such as defined by an interpretation function Φ.

$$\begin{array}{lcl}
\mathcal{R} & : & Rule \rightarrow Env \rightarrow Node \rightarrow Bool \\[1ex]
\mathcal{R}[\![\vec{p} \rightarrow \vec{p}']\!]_x^{\phi} & = & \text{let } \phi' = \mathcal{M}[\![\vec{p}]\!]_x^{\phi} \text{ in} \\
 & & \text{if } \phi' \neq \emptyset \text{ then } (\textbf{do } \Phi(\phi', p, p'); \textbf{true}) \\
 & & \text{else } \textbf{false}
\end{array}$$

We now define the matching, the tree rewriting operations and their construction by function Φ

3.1 Matching and filtering

Wadler first defined XPath matching formally [8] as a boolean function, true if the selection is not empty. Actually we need more than matching, since rewriting tree operations require node identification (*filtering*). Indeed, we have to designate the nodes we want to remove or to bind to other nodes.

Our idea is to mark the various nodes involved in the selection path with a designation tag, i.e. a $p[?v]$ qualifier, which is a sugar for $p[. == \$v]$, v being an "existential variable", not defined in the current ϕ environment. For instance, if the following expression

Example 1.

$$\vec{P} = table/tr[?a]/td[?b][2]$$

matches in environment ϕ and context node x, a new environment (a substitution) ϕ' will be established. In that case, the corresponding *tr* and *td* element identifiers x_1, x_2 will be designated by variables $\$a$ and $\$b$:

$$\phi' = \{a = x_1,\ b = x_2\}$$

We now propose to define the matching function through a translation Ξ of p terms into a set of containment constraints. A matching is found when a solution to the sytem is found. The terms of this equation have a selection semantics in conformance with the definition of section 2.2. The small example 1 above is translated into

$$\Xi(\vec{P}) = \left\{ \begin{array}{lcl} \$a & \leq & table/tr \\ \$b & \leq & \$a/td[\mathbf{position}() = 2] \end{array} \right.$$

Another way to present this translation is to generate a unique equivalent XPath term:

$$\Xi(\vec{P}) = .[\$a \sqsubseteq table/tr][\$b \sqsubseteq \$a/td[\mathbf{position}() = 2]]$$

which should select the context node in a suitable environment ϕ'. This translation [4] allows us to define the matching semantics in a quite elegant way, since it relates directly this operation to the fundamental XPath selection mechanism.

Definition 1. *Matching of a rule pattern.*

$$\begin{array}{lcl}
\mathcal{M} & : & Env \rightarrow Pattern \rightarrow Node \rightarrow Env \\[1ex]
\mathcal{M}[\![\vec{p}]\!]_x^{\phi} & = & \textit{if } \exists \phi' \mid \mathcal{S}[\![\ \Xi(\vec{p})\ ]\!]_x^{\phi'} = \{x\} \\
 & & \textit{then } \phi' \textit{ else } \emptyset
\end{array}$$

[4]We consider that the definition of Ξ doesn't present technical difficulties, and will not be detailed here.

Many solutions may exist, and moreover, they can be totally ordered through $\ll$, the document order. In order to apply a matching rule, it makes sense to choose the first solution as the default one, since it eliminates non-determinism.

3.2 Operations on the document tree

We propose now a restricted set of primitive operations used for updating the tree. We define informally the semantics of these operations in the figure 4, through an "object-oriented" style, x being the node on which the methods are invoked (we could map these definitions into DOM [4] operations, but it would be probably less compact). Note that

operation	arguments	meaning
x_i =element()		create element x_i
x_i =attribute()		create attribute x_i
x_i =text(s)	content	create text x_i
x_i =comment(s)	content	create comment x_i
x_i =pi(t,i)	target,inst.	create processing instr. x_i
x.unbind()		detach x from his parent
x.bind(x_1)	the parent	let x be child of x_1
x.name(s)	the name	set element/attr. name
x.before(x_1)	a node	set x's position before x_1
x.after(x_1)	a node	set x's position after x_1

Figure 4: Basic operations on tree

there is no explicit deletion of nodes. We consider that tree fragments which are let unbound to the tree after a rule application will be freed from memory by an independant garbage collector.

3.3 Computation of rewriting operations

Rewriting operations deduced from a given rule are expressed as a sequence of operations described in figure 4. We outline below the Φ function principles (*lhs* and *rhs* respectively stand for *left hand side* and *right hand side*).

Definition 2. *The Φ generation function: intentional definition*

1. *tree nodes designated by variables in the lhs, and not used in the rhs are deleted.*
2. *tree nodes designated by variable in the rhs, and not used in the lhs are created*[5]*.*
3. *nodes defined in the rhs are created unless they are reused from the lhs*[5]*.*
4. *nodes of the right hand side are reorganized in such a way that they will finally match the rhs expression.*

We tried to capture the essence of our rule semantics through the definition above. However, a more detailled definition is proposed hereafter:

Definition 3. *The Φ generation function: a more comprehensive definition*

1. *nodes designated by variables in the lhs, and not used in the rhs are deleted (the link to the parent node is deleted through the* unbind *operation)*
2. *nodes designated by variable in the rhs, and not used in the lhs are created* [5]
3. *nodes defined in the rhs and not designated by any variable are created* [5]
4. *nodes designated both in lhs and rhs, and whose connexity is modified by the new arrangement are let unbound*
5. *newly created nodes (or nodes covered by item 4 are bound to the tree according to connexity information of the right hand expression*
6. *nodes are named in accordance with the rhs*
7. *position of nodes are set in accordance with position information contained in the rhs*

[5]in accordance with the node type specified by the XPath expression.

As an illustration, we provide the reader with the translation of operations from right hand sides of examples 3 and 2 (see section 4). These operations are to be executed in the context ϕ' built from matching the left hand side, and with the context node x.

$\phi'(\$a).unbind()$
$\phi'(\$b).unbind()$
$x_1 = element()$
$x_1.name(\text{"apply"})$
$x_1.bind(x)$
$x_2 = element()$
$x_2.name(\text{"root"})$
$x_2.bind(x_1)$
$x_3 = element()$
$x_3.name(\text{"degree"})$
$x_4 = text(\text{"2"})$
$x_4.bind(x_3)$
$x_3.bind(x_1)$
$\phi'(\$c).unbind()$
$\phi'(\$c).bind(x_1)$

$x_3.after(x_2)$
$\phi'(\$c).after(x_3)$

Operations of example 3

$x_1 = element()$
$x_1.name(\text{"degree"})$
$x_1.bind(\phi'(\$a))$
$x_2 = text(\text{"2"})$
$x_2.bind(x_1)$
$x_1.after(\phi'(\$b))$

Operations of example 2

The reader may note that many issues remain open (we will review some of them in the conclusion), and are part of future work. We would like to design an exhaustive translation, simple enough to enable formal treatment and characterization.

3.4 Well-formedness of rules

We first propose to force syntactical restrictions on left and right terms: **not** p qualifiers and $p|p$ expressions must not contain free variables. This respectively addresses the problem of (i) introducing variables that possibly cannot be defined after matching (e.g. **not** $p[?a]$) and (ii) non-determinism in rule application (we avoid such cases at this point).

Beyond these basic preliminar restrictions, other rule specification could lead to fundamental problems. We propose to detail now all such possibilities. Our underlying method is based on two lines: (i) analyzing hypothesis related to tree connexity and node type constraints and (ii) analyzing illegal operations that may be generated. We summarize these problems below :

1. illegal node type conversions. It can be illustrated by the following ill-formed rules:

$$table[?a] \rightarrow comment()[?a]$$

which would require changing the type of a node, thus conduct to a violation of our tree data model, since a *comment* node could get elements as children. Morover, this situation simply cannot be translated into our basic tree operations, since we do not propose the corresponding primitives

2. illegal node binding. This comes with rules like
$$\mathit{table}[?a][@\mathit{border}[?b]] \rightarrow @\mathit{border}[?b]/\mathit{table}[?a]$$
3. invalid XPath expression. Terms are considered invalid when they always evaluate to the empty set $\emptyset$, such as
$$\mathit{table}[\textbf{not}\ \mathit{tr}]/\mathit{tr} \rightarrow .$$
4. multiply defined filter variables. In the following example, no solution can be found, since nodes designated by b are always distinct.
$$\mathit{table}[?a]/\mathit{tr}[?b]/\mathit{td}[?b] \rightarrow \mathit{table}[?a]$$

The first point requires a straightforward type analysis (it consists in annotating the variables with their node type, and checking the type invariance). An attractive solution to the second and third point is using a normalization process, such as described in [14]. This approach allows reducing any ill-formed XPath expressions into $\perp$, the always void XPath term[6]. Moreover, it reduces the syntactic complexity and eases the definition of subsequent processing. The forth case requires a quite simple static analysis verifying that no filtering variables are introduced more than once. Note also that such a case might be detected through the normalization process we described above, provided the containment inference system is complete, which is not yet known ([14]; the completness property is mathematically expressed, but not proved).

4. REWRITING EXAMPLES

Although the paper does not focus on rule application models, one can consider standard approaches such as leftmost-innermost strategies, were the tree is explored node by node, and rules are tried in order. In any case, the transformation terminates when no rule can by applied to any node. We found it interesting to evaluate the expressiveness of our approach against "real world" examples, i.e MathML normalization and transformation ([6, 7]). Normalization is useful in order to eliminate assumptions about default values. For instance the specification says that the *degree* argument of *root* function is 2 if not specified. That is, in content semantics,
$$\mathit{root}(a) = \mathit{root}(2, a)$$
and also in presentation semantics
$$\sqrt{a} = \sqrt[2]{a}$$
The transformation is structurally described as
$$\begin{array}{lcl}
\begin{array}{l}
\langle \mathit{apply}\rangle \\
\quad \langle \mathit{root}/\rangle \\
\quad \langle \mathit{ci}\rangle a\langle /\mathit{ci}\rangle \\
\langle /\mathit{apply}\rangle
\end{array}
& \rightarrow &
\begin{array}{l}
\langle \mathit{apply}\rangle \\
\quad \langle \mathit{root}/\rangle \\
\quad \langle \mathit{degree}\rangle 2\langle /\mathit{degree}\rangle \\
\quad \langle \mathit{ci}\rangle a\langle /\mathit{ci}\rangle \\
\langle /\mathit{apply}\rangle
\end{array}
\end{array}$$
This is expressed as

Example 2.
$$\begin{array}{rl}
& \mathit{apply}[?a][\textbf{not}\ \mathit{degree}]/\mathit{root}[?b] \\
\rightarrow & \mathit{apply}[?a]/\mathit{root}[?b]/\mathit{f\text{-}s}::*[1][\mathit{self}::\mathit{degree}][. = \text{"}\mathit{2}\text{"}]
\end{array}$$

Note that the right hand specifies that a new *degree* element must be created just after the original *root* element. The following solution, syntactically simpler, is computationally more expensive, as *apply* and *root* element are first deleted and then created

Example 3.
$$\begin{array}{rl}
& \mathit{apply}[?a][\textbf{not}\ \mathit{degree}]/\mathit{root}[?b]/\mathit{f\text{-}s}::*[?c] \\
\rightarrow & \mathit{apply}/\mathit{root}/\mathit{f\text{-}s}::\mathit{degree}[. = \text{"}\mathit{2}\text{"}]/\mathit{f\text{-}s}::*[?c]
\end{array}$$

MathML is split in two parts: one is oriented toward the semantics of mathematical expressions (Content MathML, or C-MathML), and the other is dedicated to the presentation of these expressions (P-MathML, see [7] for a tutorial introduction). This fascinating duality opens interesting transformation problems that we believe can be elegantly adressed through the present proposal. One of them is about transforming C-MathML into P-MathML already adressed through XLST in [15]. For instance, the functional expression
$$\mathit{power}(\mathit{add}(a, 3), 2)$$
can be represented as
$$(a + 3)^2$$
The corresponding MathML transformation is expressed as
$$\begin{array}{lcl}
\begin{array}{l}
\langle \mathit{apply}\rangle \\
\quad \langle \mathit{power}/\rangle \\
\quad \langle \mathit{apply}\rangle \\
\qquad \langle \mathit{plus}/\rangle \\
\qquad \langle \mathit{ci}\rangle a\langle /\mathit{ci}\rangle \\
\qquad \langle \mathit{cn}\rangle 3\langle /\mathit{cn}\rangle \\
\quad \langle /\mathit{apply}\rangle \\
\quad \langle \mathit{cn}\rangle 2\langle /\mathit{cn}\rangle \\
\langle /\mathit{apply}\rangle
\end{array}
& \rightarrow &
\begin{array}{l}
\langle \mathit{msup}\rangle \\
\quad \langle \mathit{mfenced}\rangle \\
\qquad \langle \mathit{mi}\rangle a\langle /\mathit{mi}\rangle \\
\qquad \langle \mathit{mo}\rangle + \langle /\mathit{mo}\rangle \\
\qquad \langle \mathit{mn}\rangle 3\langle /\mathit{mn}\rangle \\
\quad \langle /\mathit{mfenced}\rangle \\
\quad \langle \mathit{mn}\rangle 2\langle /\mathit{mn}\rangle \\
\langle /\mathit{msup}\rangle
\end{array}
\end{array}$$
A solution is proposed through the following rule set
$$\begin{array}{lrl}
(r_1) & & \mathit{apply}[?a]/\mathit{power}/\mathit{f\text{-}s}::*[?b]/\mathit{f\text{-}s}::*[?c] \\
& \rightarrow & \mathit{msup}/*[?b]/\mathit{f\text{-}s}::*[?c] \\
\\
(r_2) & & \mathit{apply}[?a][\mathit{add}]/*[?b]/\mathit{f\text{-}s}::*[?c] \\
& \rightarrow & \mathit{mfenced}/*[?b][1]/\mathit{f\text{-}s}::\mathit{mo}[. = \text{"}+\text{"}]/\mathit{f\text{-}s}::*[?c] \\
\\
(r_3) & & \mathit{ci}[?a] \rightarrow \mathit{mi}[?a] \\
\\
(r_4) & & \mathit{cn}[?a][@\mathit{type} = \text{"integer"}]/\textbf{text}()[?b] \\
& \rightarrow & \mathit{mn}/\textbf{text}()[?b]
\end{array}$$
Note that r_4 just handles integers, and we would have to add more rules to handle all kind of numbers that are allowed by the specification. For instance, the following one adresses *complex-cartesian* numbers.
$$\begin{array}{lrl}
(r_5) & & \mathit{cn}[?a][@\mathit{type} = \text{"complex-cartesian"}]/ \\
& & \textbf{text}()[?b]/\mathit{f\text{-}s}::\mathit{sep}/\mathit{f\text{-}s}::\textbf{text}()[?c] \\
& \rightarrow & \mathit{mn}/\mathit{mrow}/\mathit{mi}[\textbf{text}()[?b]]/ \\
& & \mathit{f\text{-}s}::\mathit{mo}[. = \text{"-"}]/\mathit{f\text{-}s}::\mathit{mi}[\textbf{text}()[?c]]
\end{array}$$

5. CONCLUSION AND FUTURE WORK

We believe that the two main difficulties described in the introduction are adressed by our proposal:

[6]Still it must be slightly extended to consider positional extensions

1. An elegant and simple extension to XPath has been proposed in order to identify useful nodes when applying a pattern to a tree.
2. A natural interpretation of the right hand side is proposed ; moreover, it allows deep reorganization of the tree, such as subtree deletion, node renaming, node creation and positional changes.

However, many issues remain unclear with respect to the generation of transformation operations. Some specifications may generate multiple interpretations, and the Φ algorithm will have to decide which is the best one. Let us consider the following rule

$$a/b[?v] \rightarrow a/*[?v]$$

According to our approach, any legal renaming of the $\$v$ node is legal. Which one should be considered ? From the maximal efficiency point of view, no renaming should be applied. But in a broader context were we would like to perform static type checking analysis (let us imagine we transform an instance of a particular document schema), a good renaming operation would be the one that preserves the validity of the instance (e.g. a c if the content model of a nodes defines only b or c children).

Let us consider another difficult case, were the generated node must satisfy a complex constraint:

$$a \rightarrow b[?c][* \textbf{ and } * \sqsubseteq \$c/d]$$

After matching, this rule has to generate a node b child of the context node, and at least a node d child of b. So it looks like that in some cases, the tree operations must satisfy complex constraints, and it is yet unclear to the author how to deal with those issues in a simple way. Part of our future work will be thus to clarify this point and to consider existing litterature on this topic, e.g. [2].

Another important issue is to relate pattern unification to pattern matching. For instance, the following two patterns shoud be considered as equivalent, up to a variable substitution

$$a[?v]/b/c[?w] \sim a[?u][b/c[?r]]$$

One way to explore is to extend XPath containment to pattern containment.

Proposition 1. *pattern containment*

$$\vec{p_1} \leq \vec{p_2} \quad \textit{iff} \quad \exists\gamma \textit{ such that } \gamma \vdash \Xi(\vec{p_1}) \leq \Xi(\vec{p_2})$$

Proposition 2. *pattern equivalence*

$$\vec{p_1} \sim \vec{p_2} \quad \textit{iff} \quad (\vec{p_1} \leq \vec{p_2} \wedge \vec{p_2} \leq \vec{p_1})$$

For the example above, a substitution could be $\gamma = \{v = \$u, w = \$r, u = \$v, r = \$w\}$

This work on rewriting will be the building block of a new XML transformation language called Ω, curently studied in the WAM project [17]. The Ω approach tries to associate such transformation rules (a bit more complex, of form $l, r \rightarrow r'$ where l is matching against an input document, and $r \rightarrow r'$ rewrites the output document) with explicit strategies. The idea is to control the rule application and allow complex transformation models while maintaining good properties with respect to type checking and static analysis. We also explore through Ω the possibility of defining non-deterministic transformation strategies, were the output document might be constructed through backtracking and some kind of constraint solving.

6. ACKNOWLEDGMENTS

We would like to thanks the many people participating to this reflexion around XPath and supporting the ongoing work on Ω transformation language: Vincent Quint and all his team from the WAM [17] INRIA project (especially Nabil Layaïda and Pierre Genevès); Jerôme Euzenat, Lionel Villard and Jean-Marc Andreoli.

7. REFERENCES

[1] N. Dershowitz, J.-P. Jouanaud *"Handbook of Theoretical Computer Science"*, Chapter on Rewriting, pp. 243-320 `Elsevier`, 1990.

[2] H. Common, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison and M.Tommasi *"Tree Automata Techniques and Applications"*, Technical Report, `www.grappa.univ-lille3.fr/tata`, October, 1999.

[3] XML Path Language (XPath) 1.0, W3C recommendation, 16 November, 1999.

[4] Document Object Model (DOM) 1.0, W3C recommendation, October, 1998.

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

[6] Mathematical Markup Language (MathML) 2.0, W3C Recommendation, 21 February, 2001.

[7] Michael Kohlhase, *New MathML tutorial*, W3C working group, `www.w3.org/Math/Documents/mathml-tutorial.pdf`, 11 March, 2003.

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

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

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

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

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

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

[14] Jean-Yves Vion-Dury and Nabil Layaïda, *"Containment of XPath Expressions: an Inference and Rewriting based approach "*, Extreme Markup Languages, Montréal August, 2003. `http://wam.inrialpes.fr/publications /2003/xtrem2003/xtrem2003.pdf`

[15] Emmanuel Piétriga, *MathMLc2p*, XSLT based translation of content MathML into presentation MathML, April, 2000. `http://opera.inrialpes.fr/people/ Emmanuel.Pietriga/mathml2cp.html`

[16] Wolfgang May *"XPath-Logic and PathLog: A Logic-Based Approach for Declarative XML Data Manipulation"*, Technical Report No. 149, Institut für Informatik, Freiburg, Germany Feb, 2001.

[17] WAM *"Web Adaptation and Multimedia"*, Research project from INRIA Rhône-Alpes, `http://wam.inrialpes.fr`.