Calculating Tree Navigation with Symmetric Relational Zipper

Yuta IKEDA   Susumu NISHIMURA

Department of Mathematics, Faculty of Science
Kyoto University
Kyoto 606-8502, Japan
{yuta-28,susumu}@math.kyoto-u.ac.jp

Abstract

Navigating through tree structures is a core operation in tree processing programs. Most notably, XML processing programs intensively use XPath, the path specification language that locates particular nodes in a given document structure.

This paper develops a theory for reasoning about equalities of tree navigation programs. In functional programming languages, tree navigation operations can be cleanly implemented as functions operating over the data structure known as Huet's zipper. The tree navigation functions are expected to have certain nice symmetric properties (e.g., a function going one-level down in the tree structure would be the inverse of another function coming back the other way around, and vice versa), but they are not indeed a perfect symmetry, due to partiality and non-injectivity of the functions.

In order to fully exploit the symmetry indicated by tree navigation operations, we model them by relations, instead of functions. The relational specification allows us to derive useful equations by simple calculations. We apply the calculational method to derive certain equalities of XPath expressions. The point-free relational reasoning not only leads to a concise justification of some known results but also establishes equations for a larger class of tree navigation operations, including those specified with negative constraints and those beyond XPath expressibility.

Categories and Subject Descriptors F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Algebraic approaches to semantics; D.1.1 [*Programming Techniques*]: Applicative (Functional) Programming

General Terms Languages, Theory

Keywords Tree navigation, Zipper, XML path language, Relational calculus, Algebra of programming

1. Introduction

Huet's zipper [9] provides us a way to elegantly implement various operations for manipulating tree structures, including tree navigation, in functional programming languages. A zipper is a pair of a binary tree and a context, as specified below in the O'Caml syntax.[1]

```
type 'a tree =
  Leaf | Node of 'a tree * 'a * 'a tree
type 'a path =
    Top
  | L of 'a * 'a tree * 'a path
  | R of 'a * 'a tree * 'a path
type 'a zipper = 'a tree * 'a path
```

The context information is represented by a list-like structure, which records, in the reverse order, the path by which the current node is reached from the root. The constructor $\mathtt{L}(i,r,p)$ corresponds to a single traversal along a left branch of a tree node $\mathtt{Node}(l,i,r)$, where p is the path to reach the tree node from the root. The constructor $\mathtt{R}(i,l,p)$ has a role that simply interchanges the left and the right. The constructor `Top` stands for an empty path, indicating the root position of the whole tree. In Figure 1, we give four basic tree navigation functions on zippers. The function dn_L (resp., dn_R) descends the binary tree structure down by one level along a left (resp., right) branch, if possible. Conversely, the function up_L (resp., up_R) returns to its parent node by climbing up a left (resp., right) branch by one level, if possible.

The above definitions of tree navigation over zippers indicate that certain natural symmetric properties would hold: `dnL` and `upL` are inverses of each other and so are `dnR` and `upR`. This symmetry indicates that there could be many chances of simplification and optimization in tree navigation programs. We would be able to eliminate any symmetric pair of functions which are statically recognized as being executed successively.

The symmetry argued above, however, is not as perfect as one might expect. First, the above basic tree navigation operations and many other more complex ones as well are partial functions and hence a composition of a (seemingly) symmetric pair would work as a neutral operation only for limited inputs. (E.g., $\mathtt{dnL} \circ \mathtt{upL}$ is a *partial* identity.) Second, not every tree navigation operation necessarily has a corresponding inverse operation. For example, consider the following tree navigation function that climbs up any (left or right) branch by one level, if possible.

[1] The choice of the implementation language is rather a matter of tastes and the results in this paper applies to any other suitable functional language such as Haskell, as long as trees and contexts involved are assumed to have finite structures. Although we will only consider binary trees in this paper, the fundamental techniques should successfully apply to zippers for other data strucutres.

©ACM, 2011. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version is to appear in the proceedings of ACM SIGPLAN 2011 Workshop on Partial Evaluation and Program Manipulation (PEPM'11), January 24–25, 2011, Austin, Texas, USA.

```
let dnL = function
  | (Node(l,i,r),p) -> (l, L(i,r,p))
  | _ -> failwith "no way for dnL"

let upL = function
  | (l,L(i,r,p)) -> (Node(l,i,r),p)
  | _ -> failwith "no way for upL"

let dnR = function
  | (Node(l,i,r),p) -> (r, R(i,l,p))
  | _ -> failwith "no way for dnR"

let upR = function
  | (r,R(i,l,p)) -> (Node(l,i,r),p)
  | _ -> failwith "no way for upR"
```

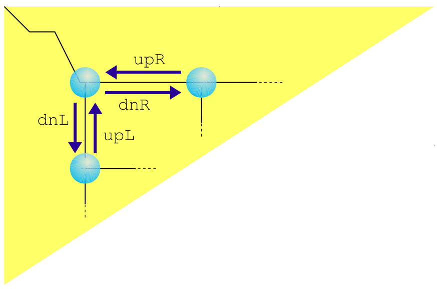

Figure 1. Tree navigation functions and a pictorial image of their operations. Left branches are drawn by vertical edges and right branches are drawn by horizontal edges.

```
let go_up = function
  | (l,L(i,r,p)) -> (Node(l,i,r),p)
  | (r,R(i,l,p)) -> (Node(l,i,r),p)
  | _ -> failwith "no way for go_up"
```

This function is not invertible, because it is not injective.

The aim of this paper is to develop a theory of tree navigation operations with perfect symmetric properties. For this, we leave the realm of functional programming paradigm and move to that of relational one [4]. The issue of partiality and non-injectivity discussed above is avoided by a relational modeling of zipper operations. By the relational modeling, the above mentioned basic navigation primitives have a perfect symmetry via relational converse, instead of functional inverse. A variety of tree navigation programs can be specified by combining these basic symmetric operations with a few composition operations on relations.

We also develop a set of algebraic laws that are useful for deriving the properties of relationally specified tree navigation. In order to demonstrate how our theory can be used to formally establish the properties of tree navigation operations via simple calculation, we consider equalities of given XPath [3] expressions that locate tree nodes in XML structured document trees. We will give a relational interpretation of XPath expressions, by which a range of equalities is instantly obtained by the symmetry intrinsic to our relational modeling. A more involved equalities on XPath predicates, which are an XPath construct that imposes further confinement to the set of located nodes, are also discussed both for positive (i.e., negation free) and negative fragments.

In the course of relational reasoning on XPath expressions, we introduce a new relational operation, called symmetric closure (Section 3), and develop several laws for calculating with that closure. This new closure operation is a key to the relational modeling of XPath expressions involving predicates (even including negatives) and the effective derivation of equalities on those XPath expressions.

1.1 Related work

The results presented in this paper is built on the foundation of the relational calculus (or algebra of programming, in other words) [4]. The point-free style calculation contributes to simplified derivations of equalities that we develop in later sections. It would be much harder to justify the same results in the pointwise style.

There are several studies that develop a set of laws for equational reasoning on XPath (or its relatives such as XQuery) [5, 6, 12, 13]. Each provides a particular set of reasoning laws for solving the problem in concern and henceforth the applicability of each proposal varies, depending on the particular choice of laws. Che et al. [5] proposed to use PAT-algebra and Cunha et al. [6] provided a set of laws for algebraic transformation of structure-shy programs, but both do not deal with reverse axes (i.e., upward tree navigation).

Olteanu et al. have extensively studied the laws for eliminating reverse axes [12, 13]. They provided an elegant set of equalities that exploit the symmetry we mentioned earlier. However, they did not give any equality for those XPath expressions that contain negative predicates. In Section 5, we will show that some XPath expression with negatives can be equated to a tree navigation specification which cannot be expressed in XPath language but in our relational language. This is due to our finer modeling of tree navigation via relational composition of a few navigation primitives, none of which cannot be expressed in XPath language. The finer granularity provides more opportunities for developing a larger set of equalities. Furthermore, our relational modeling allows point-free derivations that contribute to a simpler calculational reasoning. Some of simple equalities given in [12, 13] is an immediate consequence of relational converse. Also, the point-free style of derivations are more beneficial for reasoning about negative predicates. It would be hard and tend to be unmanageable to formally conduct justification of equalities on negatives in the pointwise style as in [12].

Genevès, et al. [8] provides an automatic procedure for various XPath decision problems, based on a modal μ-calculus. Though having different interpretation domains, theirs and ours are quite similar in their way of modeling tree navigation. Genevès, et al. encode tree navigation by existential modalities, which correspond to our tree navigation primitives. It should be however noted that they restrict their formulas to cycle-free fragments, which have no possible infinite iterations of a modality and its reverse. Though the cycle-free restriction is needed for designing an automatic decision procedure, the restriction does not matter for their development, as every XPath specification can be expressed by a cycle-free formula. When we do not care about automation and go beyond the XPath language, we claim that this restriction seems a bit restrictive. As we develop in later sections, a less strict well-foundedness condition is sufficient for our calculation. Nevertheless it is quite interesting that different formalizations via modal μ-calculus and relational calculus exhibit many similarities. A thorough comparison on this point would merit further investigation.

Outline. The rest of the paper is organized as follows. We first present our relational zipper operations that have symmetric properties, together with some basics on relational calculus in Section 2. Section 3 gives a relational semantics of XPath expressions, where a new closure operator is introduced. Thereafter we discuss XPath equalities for negation-free expressions in Section 4 and negative expressions in Section 5. Section 6 briefly describes the translation method for implementing relations in functional languages. Finally Section 7 concludes the paper.

2. Symmetric relational zipper operations

We introduce basic zipper operations that have symmetric properties. This section also contains a brief introduction to the relational calculus. For a more detailed introduction to this field, see standard references, e.g. [2, 4].

2.1 Relations

Given arbitrary sets A and B, a binary relation $R : A \leftarrow B$ is a subset of Cartesian product $A \times B$. We say R relates the input $b \in B$ to the output $a \in A$ if $(a,b) \in R$, written $a\,R\,b$. The relations of type $A \leftarrow B$ forms a complete lattice with $\subseteq$ as the ordering, where $\cup$ and $\cap$ are the join and meet operators, respectively. The largest relation in the lattice is the total relation, written $\Pi : A \leftarrow B$, and the smallest relation is the empty relation, written $\emptyset : A \leftarrow B$.

Two relations $R : A \leftarrow B$ and $S : B \leftarrow C$ can be composed into a single relation, written $R \circ S$, which is defined by $a(R \circ S)c \equiv \exists b \in B : a\,R\,b \wedge b\,S\,c$. The relational composition is associative and has the identity relation id as a unit element, where $a\ id\ b \equiv a = b$. It is also monotonic w.r.t. the ordering $\subseteq$, i.e., $R \circ S \subseteq R' \circ S'$ if $R \subseteq R'$ and $S \subseteq S'$. The relational composition distributes over the join (i.e., $R \circ (S \cup T) = (R \circ S) \cup (R \circ T)$), but it only semi-distributes over the meet (i.e., $R \circ (S \cap T) \subseteq (R \circ S) \cap (R \circ T)$).

A converse relation of $R : A \leftarrow B$, written $R^{\smile}$, is a relation that exchanges the inputs and outputs of R, that is, $b\,R^{\smile}\,a \equiv a\,R\,b$. The converse is self-dual (i.e., $R^{\smile\smile} = R$) and distributes over the join and meet (i.e., $(S \cup T)^{\smile} = S^{\smile} \cup T^{\smile}$ and $(S \cap T)^{\smile} = S^{\smile} \cap T^{\smile}$). We also have equality $(R \circ S)^{\smile} = S^{\smile} \circ R^{\smile}$ and the so called *modular law*: $(R \circ S) \cap T \subseteq (R \cap (T \circ S^{\smile})) \circ S$.

A relation $R : B \leftarrow A$ is called *simple* (resp., *injective*) if $R \circ R^{\smile} \subseteq id$. (resp., $R^{\smile} \circ R \subseteq id$). Dually, R is called *entire* (resp., *surjective*) if $R^{\smile} \circ R \supseteq id$. (resp., $R \circ R^{\smile} \supseteq id$).

A relation $R : A \leftarrow A$ is called *coreflexive* if $R \subseteq id$. Coreflexive relations are often used for checking membership of values. We write $C?$ for the coreflexive relation induced by a set C, that is, $x C? y \equiv x = y \wedge x \in C$. Thus a composed relation, say, $R \circ C?$ confines the input of R to the values that belong to C. Coreflexives are idempotent and commutative for relational composition. That is, $R \circ R = R$ and $R \circ S = S \circ R$ for any coreflexives $R, S : A \leftarrow A$.

Given two relations $R : A \leftarrow C$ and $S : B \leftarrow D$, we define a *relational product* $R \times S : (A \times B) \leftarrow (C \times D)$ by $(a,b)\ R \times S\ (c,d) \equiv a\,R\,c \wedge b\,S\,d$.

It is often convenient and instructive to characterize certain relational operators via Galois connection [2]. A Galois connection, in the present relational setting, is identified by a pair of operations f and g (called lower and upper adjoints, respectively) over relations that satisfy the equivalence $f(R) \subseteq S \equiv R \subseteq g(S)$ for any relations R and S.

Here we introduce two relational operators characterized by Galois connections, the relational difference operator $(-S)$ and the domain operator dom:

$$R - S \subseteq T \equiv R \subseteq S \cup T \qquad \text{(GC-DIFF)}$$

$$dom\,R \subseteq S \equiv R \subseteq \Pi \circ S, \quad \text{for all coreflexive } S. \qquad \text{(GC-DOM)}$$

The relational difference operator $(-S)$ is the left adjoint of the operator $(S\cup)$. The difference operator corresponds to the set difference, that is, $a\ (R - S)\ b$ if and only if $a\,R\,b$ but not $a\,S\,b$.

The domain operator dom is the left adjoint of the right conditional operator $(\Pi\circ)$. An explicit definition is given by $dom\ R = id \cap R^{\smile} \circ R$. Thus, for any relation $R : B \leftarrow A$, $dom\ R : A \leftarrow A$ is a coreflexive representing the set of inputs that have at least a single corresponding output.

By the Knaster-Tarski theorem, if f is a monotonic mapping on relations (i.e., $f(R) \subseteq f(S)$ whenever $R \subseteq S$), there exist the least fixpoint $\mu X.f(X)$ and the greatest fixpoint $\nu X.f(X)$, both of which are solutions of equation $X = f(X)$. The least and greatest fixpoints have the following properties useful for relational reasoning.[2, 10]

$$\mu X.f(X) = f(\mu X.f(X)) \qquad (\mu\text{-computation})$$

$$\mu X.f(X) \subseteq R \Leftarrow f(R) \subseteq R \qquad (\mu\text{-induction})$$

$$\nu X.f(X) = f(\nu X.f(X)) \qquad (\nu\text{-computation})$$

$$R \subseteq \nu X.f(X) \Leftarrow R \subseteq f(R) \qquad (\nu\text{-induction})$$

In particular, the reflexive transitive closure of a relation R is the least fixpoint $\mu X.id \cup R \circ X$, which we write R^* for short. The closure R^* satisfies the equations $S \circ R^* = \mu X.(S \cup X \circ R)$ and $R^* \circ S = \mu X.(S \cup R \circ X)$. Hence $R^* = \mu X.(id \cup X \circ R)$ and $R^* \circ R = R \circ R^*$ also hold. We will often write R^+ for $R \circ R^*$.

2.2 Relational zipper operations

In the rest of the paper, we assume the zipper data structure that we have given in Introduction. We also express the type of zippers by Zipper, rather than by `'a zipper` in the concrete O'Caml syntax, obfuscating the parametrized type `'a` of data stored in the tree nodes.

We first introduce a pair of relations representing generic operations for navigating up and down trees by one level, namely, the relations Dn : Zipper $\leftarrow$ Zipper and Up : Zipper $\leftarrow$ Zipper, which are defined as the smallest relations satisfying the following properties for any i, l, r, and p of appropriate types.

$$(l, \mathtt{L}(i,r,p))\ Dn\ (\mathtt{Node}(l,i,r),p)$$

$$(r, \mathtt{R}(i,l,p))\ Dn\ (\mathtt{Node}(l,i,r),p)$$

$$(\mathtt{Node}(l,i,r),p)\ Up\ (l, \mathtt{L}(i,r,p))$$

$$(\mathtt{Node}(l,i,r),p)\ Up\ (r, \mathtt{R}(i,l,p))$$

By definition, Up and Dn are a pair of symmetric operations, in the sense that they are converses of each other. It should be however noted that they are not inverses of each other. Neither $Up \circ Dn$ nor $Dn \circ Up$ is equal to id; We can only say that $Up \circ Dn = \mathtt{Node}? \times id \subseteq id$.

The four primitive operations introduced in Introduction can be defined in the relational setting as follows.

$$dn_L = (id \times \mathtt{L}?) \circ Dn \qquad dn_R = (id \times \mathtt{R}?) \circ Dn$$

$$up_L = Up \circ (id \times \mathtt{L}?) \qquad up_R = Up \circ (id \times \mathtt{R}?)$$

In abuse of notations, we use each constructor name `C` of a datatype to denote the set of values whose outermost constructor is `C`. E.g., `L` represents the set of all the paths in the form $\mathtt{L}(i,r,p)$. Thus `C`? denotes the coreflexive relation that checks if the outermost constructor is `C`. Hence dn_L and dn_R restrict the more general relation Dn to the navigation of the appropriate direction downward; up_L and up_R can move upward only if the direction moving upward matches that of the most recent downward navigation.

The four navigation operations exhibit clean symmetric properties, as we shall show below.

Each pair of dn and up of the same subscript are the converse of each other, i.e.,

$$dn_L{}^{\smile} = up_L \qquad dn_R{}^{\smile} = up_R \qquad \text{(UPDN-CONVERSE)}$$

Furthermore the composition of each converse pair is coreflexive.

$$dn_L \circ up_L \subseteq id \qquad up_L \circ dn_L \subseteq id$$

$$dn_R \circ up_R \subseteq id \qquad up_R \circ dn_R \subseteq id \qquad \text{(DNUP-COREFL)}$$

In other words, they are all simple and injective relations.

On the other hand, certain pairs of different subscripts induce empty relations.

$$up_L \circ dn_R = \emptyset \qquad up_R \circ dn_L = \emptyset \qquad \text{(UPDN-EMP)}$$

e ::=	XPath
$/p$	absolute path
$\mid p$	relative path
$\mid e \cup e$	union
$\mid e \cap e$	intersection
p ::=	Path
$a :: *$	step
$\mid a :: \beta$	step with node test
$\mid p[q]$	predicate
$\mid p/p$	path composition
a ::=	Axis
self |	
| child | foll-sibling | desc | desc-or-self | following	
| parent | prec-sibling | anc | anc-or-self | preceding	
q ::=	Predicate
q and q	conjunction
| q or q	disjunction
| not q	negation
| p	path

Figure 2. XPath expression syntax

We notice that not every similar pair of relations necessarily induces an empty relation, e.g., $dn_R \circ up_L \neq \emptyset$.

From the fact that coreflexives derived from distinct constructors apparently have no intersection, e.g., $\texttt{L?} \circ \texttt{Top?} = \emptyset$, we also have additional pairs of relations that induce empty relations.

$$up_L \circ (id \times \texttt{Top?}) = \emptyset \qquad up_R \circ (id \times \texttt{Top?}) = \emptyset \qquad (\textsc{Up-Emp})$$
$$dn_L \circ (\texttt{Leaf?} \times id) = \emptyset \qquad dn_R \circ (\texttt{Leaf?} \times id) = \emptyset \qquad (\textsc{Dn-Emp})$$

We can also induce several properties:

$$\begin{aligned} S \subseteq T \circ \rho &\implies S \circ \rho^{\smile} \subseteq T \\ S \subseteq \rho \circ T &\implies \rho^{\smile} \circ S \subseteq T \end{aligned} \qquad (\textsc{UpDn-Shunt})$$
$$\rho = \rho \circ \rho^{\smile} \circ \rho \qquad (\textsc{UpDn-Triple})$$

where ρ is either of dn_L, up_L, dn_R, or up_R.

3. Relational interpretation of XPath expressions

We give a relational modeling of tree navigation expressed by a core XPath language[2], whose syntax is given in Figure 2.

In Figure 3, we give the relational semantics of XPath expressions, in the style of denotational semantics [14].

An XPath expression e is either a relative path, an absolute path, or unions and intersections of them. Its formal meaning is given by a relation, written $I_e[\![e]\!]$, that relates each input (namely, a zipper recording the current tree node and context) to the (output) nodes that are located by e. A relative path p locates tree nodes reached from the current node via p, while an absolute path $/p$ locates tree nodes reached via p from the root node. The interpretation of the absolute path is thus appended with $(id \times \texttt{Top?}) \circ (up_L \cup up_R)^*$ that navigates tree context to the root.

Step expressions $a :: *$ and $a :: \beta$ are a path that locates all nodes reached via the axis a. The former does not care about node labels but the latter locates only those β-labeled nodes. (In the rest of the paper, we will use small Greek letters for node labels.) The name of a node label is examined by a coreflexive $\beta?$, which is

[2] Some axis specifiers are shortened (descendant as desc; ancestor as anc). Also, XML attributes are omitted for the sake of simplicity, although they can be easily supported by enriching node tags with additional attribute information.

I_e : XPath $\rightarrow$ (Zipper $\leftarrow$ Zipper)

$$\begin{aligned} I_e[\![/p]\!] &= I_p[\![p]\!] \circ (id \times \texttt{Top?}) \circ (up_L \cup up_R)^* \\ I_e[\![p]\!] &= I_p[\![p]\!] \\ I_e[\![e_1 \cup e_2]\!] &= I_e[\![e_1]\!] \cup I_e[\![e_2]\!] \\ I_e[\![e_1 \cap e_2]\!] &= I_e[\![e_1]\!] \cap I_e[\![e_2]\!] \end{aligned}$$

I_p : Path $\rightarrow$ (Zipper $\leftarrow$ Zipper)

$$\begin{aligned} I_p[\![a :: *]\!] &= (\texttt{Node?} \times id) \circ I_a[\![a]\!] \\ I_p[\![a :: \beta]\!] &= \beta? \circ I_a[\![a]\!] \\ I_p[\![p_1/p_2]\!] &= I_p[\![p_2]\!] \circ I_p[\![p_1]\!] \\ I_p[\![p[q]]\!] &= I_q[\![q]\!] \circ I_p[\![p]\!] \end{aligned}$$

I_a : Axis $\rightarrow$ (Zipper $\leftarrow$ Zipper)

$$\begin{aligned} I_a[\![\text{self}]\!] &= id \\ I_a[\![\text{child}]\!] &= dn_R^* \circ dn_L & I_a[\![\text{parent}]\!] &= up_L \circ up_R^* \\ I_a[\![\text{foll-sibling}]\!] &= dn_R^+ & I_a[\![\text{prec-sibling}]\!] &= up_R^+ \\ I_a[\![\text{desc}]\!] &= (dn_R^* \circ dn_L)^+ & I_a[\![\text{anc}]\!] &= (up_L \circ up_R^*)^+ \\ I_a[\![\text{desc-or-self}]\!] &= (dn_R^* \circ dn_L)^* & I_a[\![\text{anc-or-self}]\!] &= (up_L \circ up_R^*)^* \\ I_a[\![\text{following}]\!] &= (dn_R^* \circ dn_L)^* \circ dn_R^+ \circ (up_L \circ up_R^*)^* \\ I_a[\![\text{preceding}]\!] &= (dn_R^* \circ dn_L)^* \circ up_R^+ \circ (up_L \circ up_R^*)^* \end{aligned}$$

I_q : Predicate $\rightarrow$ (Zipper $\leftarrow$ Zipper)

$$\begin{aligned} I_q[\![q_1 \text{ and } q_2]\!] &= I_q[\![q_1]\!] \cap I_q[\![q_2]\!] \\ I_q[\![q_1 \text{ or } q_2]\!] &= I_q[\![q_1]\!] \cup I_q[\![q_2]\!] \\ I_q[\![p]\!] &= dom\ I_p[\![p]\!] \end{aligned}$$

Figure 3. Relational interpretation of XPath expressions

```
<a>
  <b></b>
  <c>
    <d></d>
    <e></e>
    <f>
      <g><h></h></g><i></i>
    </f>
    <j></j>
    <k><l></l></k>
  </c>
</a>
```

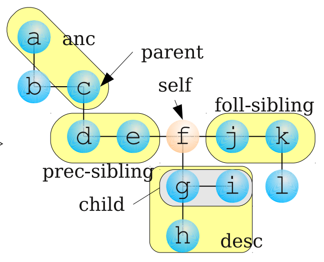

Figure 4. An XML document and its binary tree representation, with several axes relative to the `f` node

derived from the set of all zippers locating β-labeled nodes, i.e., $\{(\texttt{Node}(l,i,r),p) \in \text{Zipper} \mid i = \beta\}$.

XPath provides various axes that can be classified into subgroups. The self axis is a neutral axis that navigates the current node solely to itself. The first half of the rest of the axes, i.e., child, foll-sibling, etc., are called forward axes, as they always navigate downward. Here we notice that we model XML document structure as shown in Figure 4. Every left branch connects a parent node and its first child; Every right branch connects adjacent sibling nodes. Thus the forward axes always proceed in the document order. If a node has no child or no sibling that follows, the corresponding branch is connected with Leaf, which is omitted from the figure. The other half of the axes, i.e., parent, prec-sibling, etc., are called reverse axes, as they always navigate upward. It is easy to observe that the forward and reverse axes are connected by the symmetry, e.g, $I_a[\![\text{child}]\!] = I_a[\![\text{parent}]\!]^{\smile}$.

Paths can be arbitrarily combined for successive navigation via $p_1/p_2/\cdots/p_n$.

Example 1 Let us show an example of relational interpretation of a simple XPath expression.

$$\begin{aligned} & I_e[\![\text{parent} :: */\text{foll-sibling} :: */\text{child} :: \alpha]\!] \\ = \; & I_p[\![\text{child} :: \alpha]\!] \circ I_p[\![\text{foll-sibling} :: *]\!] \circ I_p[\![\text{parent} :: *]\!] \\ = \; & \alpha? \circ dn_R^* \circ dn_L \circ (\texttt{Node}? \times id) \circ dn_R^+ \circ (\texttt{Node}? \times id) \circ up_L \circ up_R^* \end{aligned}$$

The XPath language allows us to further restrict the node set located by a path. A predicated path $p[q]$ confines the nodes located by p to those satisfying a predicate q, which is either a relative path or conjunctions, disjunctions, or negations of predicates. (We exclude negative predicates from our discussion for the moment. We will return to this topic in Section 5.)

We interpret the confinement via predicates by coreflexive relations. Conjunctions and disjunctions are trivially interpreted by the meets and joins. Given a predicated path $p[p']$, where p' is a path, the predicated path is interpreted as a path p whose located nodes are however confined to those nodes that have at least a single tree node reached via the path p'. Thus $p[p']$ is interpreted by the relation $(dom\ I_p[\![p']\!]) \circ I_p[\![p]\!]$.

For any coreflexive relations R and S, the following properties hold.

$$dom\ R = R \qquad (\text{COREFL-DOM})$$

$$R \cup S \text{ and } R \cap S \text{ are coreflexive} \qquad (\text{COREFL-JOINMEET})$$

$$R \cap S = R \circ S = S \circ R \qquad (\text{COREFL-COMP})$$

Although we have given a formal specification of XPath expressions including predicates, it is not sufficient for effective calculation. As we have seen above, a predicate would be interpreted to a coreflexive relation of the form $dom\ R$, but its bare definitional expansion $id \cap R^{\smile} \circ R$ would not be suitable for calculation. (Recall that the meet $\cap$ operator only semi-distributes over relational compositions.) Below we develop a set of simplification laws that aid calculating predicated paths.

$$R = R \circ dom\ R \qquad (\text{DOM-DOMAIN})$$

$$\Pi \circ dom\ R = \Pi \circ R \qquad (\text{DOM-RCOND})$$

$$dom\ (R \circ S) = S^{\smile} \circ dom\ R \circ S, \quad \text{if } S \text{ is injective} \qquad (\text{DOM-COMP})$$

The laws we have given so far are an easy exercise of relational calculation, which we leave to the reader.

Composition of (injective) relations can be calculated with (DOM-COMP). For calculation of the domain of a closure operator, we introduce another closure operator, called *symmetric closure*.

Definition 3.1 (Symmetric closure)

$$(R)[S_1, S_2, \ldots, S_n]^* = \mu X. \big(R \cup \textstyle\bigcup_{i=1}^n (S_i{}^{\smile} \circ X \circ S_i)\big) \quad (n \geq 1)$$

Suppose R is coreflexive and $S_1, S_2, \ldots, S_n$ are all injective. Then it is easy to see that $(R)[S_1, S_2, \ldots, S_n]^*$ is coreflexive.

Intuitively, $(R)[S_1, S_2, \ldots, S_n]^*$ is equivalent to the infinite union of relations of the form $S_{n_1}{}^{\smile} \circ S_{n_2}{}^{\smile} \circ \ldots \circ S_{n_k}{}^{\smile} \circ R \circ S_{n_k} \circ \ldots \circ S_{n_2} \circ S_{n_1}$, where $1 \leq n_i \leq n$ for all $i \in \{1, 2, \ldots, k\}$ $(k \geq 0)$. In words, $(R)[S_1, S_2, \ldots, S_n]^*$ intends a procedure that traverses zero or more navigation steps by S_i's, makes a test on the reached node via R, and then comes back to the original position along the reverse path.

Since the reverse traversal is always successful (provided the coreflexivity and injectivity above), the right condition of the symmetric closure satisfies the following equation.

$$\Pi \circ (R)[S_1, S_2, \ldots, S_n]^* = \Pi \circ R \circ (\textstyle\bigcup_{i=1}^n S_i)^*,$$
for any coreflexive R and injective $S_1, S_2, \ldots, S_n$.
(SYMCLS-RCOND)

This equation is derived as follows. Writing U for $\bigcup_{i=1}^n S_i$, we calculate:

$$\begin{aligned} & \Pi \circ (R)[S_1, \ldots, S_n]^* \subseteq \Pi \circ R \circ U^* \\ \Leftarrow \quad & \text{monotonicity} \\ & (R)[S_1, \ldots, S_n]^* \subseteq U^{\smile *} \circ R \circ U^* \\ \Leftarrow \quad & \mu\text{-induction; distributivity} \\ & R \cup \big(U^{\smile} \circ U^{\smile *} \circ R \circ U^* \circ U\big) \subseteq U^{\smile *} \circ R \circ U^* \\ \Leftarrow \quad & \mu\text{-computation; distributivity} \\ & \textbf{true} \end{aligned}$$

The converse inclusion follows as below.

$$\begin{aligned} & \Pi \circ R \circ U^* \subseteq \Pi \circ (R)[S_1, \ldots, S_n]^* \\ \Leftarrow \quad & \mu\text{-induction} \\ & \Pi \circ R \cup \Pi \circ (R)[S_1, \ldots, S_n]^* \circ U \subseteq \Pi \circ (R)[S_1, \ldots, S_n]^* \\ \Leftarrow \quad & \mu\text{-computation; distributivity; monotonicity} \\ & \Pi \circ (R)[S_1, \ldots, S_n]^* \circ S_i \subseteq \Pi \circ S_i{}^{\smile} \circ (R)[S_1, \ldots, S_n]^* \circ S_i, \text{ for all } i \\ \equiv \quad & (\text{DOM-RCOND}); (\text{DOM-COMP}); (\text{COREFL-DOM}) \\ & \textbf{true} \end{aligned}$$
□

We use the following law to calculate $dom(R \circ (\bigcup_{i=1}^n S_i)^*)$.

$$dom(R \circ (\textstyle\bigcup_{i=1}^n S_i)^*) = (dom\ R)[S_1, S_2, \ldots, S_n]^*,$$
if $S_1, S_2, \ldots, S_n$ are all injective.
(SYMCLS-DOM)

To show this equation, we calculate:

$$\begin{aligned} & dom(R \circ (\textstyle\bigcup_{i=1}^n S_i)^*) \subseteq (dom\ R)[S_1, S_2, \ldots, S_n]^* \\ \equiv \quad & (\text{GC-DOM}) \\ & R \circ (\textstyle\bigcup_{i=1}^n S_i)^* \subseteq \Pi \circ (dom\ R)[S_1, S_2, \ldots, S_n]^* \\ \equiv \quad & (\text{SYMCLS-RCOND}) \\ & R \circ (\textstyle\bigcup_{i=1}^n S_i)^* \subseteq \Pi \circ dom\ R \circ (\textstyle\bigcup_{i=1}^n S_i)^* \\ \Leftarrow \quad & U \subseteq \Pi \circ dom\ U \text{ for any } U, \text{ by } (\text{GC-DOM}) \\ & \textbf{true} \end{aligned}$$

The converse inclusion is shown as below.

$$\begin{aligned} & (dom\ R)[S_1, S_2, \ldots, S_n]^* \subseteq dom(R \circ (\textstyle\bigcup_{i=1}^n S_i)^*) \\ \Leftarrow \quad & \mu\text{-induction; monotonicity} \\ & \textstyle\bigcup_{j=1}^n \big(S_j{}^{\smile} \circ dom(R \circ (\bigcup_{i=1}^n S_i)^*) \circ S_j\big) \subseteq dom(R \circ (\bigcup_{i=1}^n S_i)^*) \\ \equiv \quad & \text{Definition of } dom\text{; semi-distributivity} \\ & \textstyle\bigcup_{j=1}^n (S_j{}^{\smile} \circ S_j) \cap \bigcup_{j=1}^n \big(S_j{}^{\smile} \circ (\bigcup_{i=1}^n S_i)^{\smile *} \circ R^{\smile} \circ R \circ (\bigcup_{i=1}^n S_i)^* \circ S_j\big) \\ & \subseteq id \cap \big((\textstyle\bigcup_{i=1}^n S_i)^{\smile *} \circ R^{\smile} \circ R \circ (\bigcup_{i=1}^n S_i)^*\big) \\ \Leftarrow \quad & S_j\text{'s are injective; monotonicity} \\ & \textbf{true} \end{aligned}$$
□

Example 2 Let us show an interpretation of a predicated XPath expression.

$$I_e[\![\text{child} :: \beta[\text{parent} :: \alpha]]\!]$$
$=$ by definition
$$dom\ (\alpha? \circ up_L \circ up_R^*) \circ \beta? \circ dn_R^* \circ dn_L$$
$=$ (SymCls-Dom); (Dom-Comp); (Corefl-Dom)
$$(dn_L \circ \alpha? \circ up_L)[up_R]^* \circ \beta? \circ dn_R^* \circ dn_L$$

In Section 5, where we examine negative XPath predicates, we need the fact that the least fixpoint of a relational closure $S \circ R^*$, i.e., the least solution of the equation $X = S \cup X \circ R$ in X, coincides with the greatest fixpoint for a certain class of R. This is equivalent to the unique extendability property of relational closures, which has been proved by Doornbos et al. [7].

They have shown that the unique extendability property holds under the well-foundedness condition:

Definition 3.2 (well-foundedness, [7]) *We call a relation R well-founded, if $S \subseteq S \circ R$ implies $S \subseteq \emptyset$ for any relation S.*

This definition echoes the usual definition found in the order theory: There is no infinite chain $a_0, a_1, a_2, \ldots$ satisfying $a_{i+1}\ R\ a_i$ for every i.

Since we model our tree navigation by relations over the zippers of finite trees and contexts, our tree navigation primitives up_L, dn_L, etc. are apparently well-founded and so are some of their unions, e.g., $dn_L \cup dn_R$ and $up_L \cup up_R$. Note that not every union is well-founded. (For instance, $dn_L \cup up_L$ is not.)

Thus we can use the following law in calculation.

$$S \circ R^* \ = \ \nu X.(S \cup X \circ R) \qquad \text{(Closure-UEP)}$$

We can also prove that the symmetric closure also has the unique extendability property and hence the coincidence of the least and greatest fixpoints. Nevertheless we do not include this result to the present paper, as it is not needed for the development below.

4. Relational reasoning of XPath expressions

We examine how equalities of negation-free XPath expressions can be established by calculation. All the XPath equalities examined in this section are taken from [12, 13] but the proofs are much different, as we derive the equations in the point-free style.

4.1 The child/parent symmetry

Let us show the equalities of the following expressions:

child :: β[parent :: α] and self :: α/child :: β.

This is equivalent to showing the following equation over relations.

$$(dn_L \circ \alpha? \circ up_L)[up_R]^* \circ \beta? \circ dn_R^* \circ dn_L = \ \beta? \circ dn_R^* \circ dn_L \circ \alpha? \quad (4.1)$$

To show this equation, let us write Q for $(dn_L \circ \alpha? \circ up_L)[up_R]^*$. By the commutativity of coreflexives, it suffices to show the equation $Q \circ dn_R^* \circ dn_L = dn_R^* \circ dn_L \circ \alpha?$.

We first show next two lemmas.

(i) $up_R^* \circ dn_R^* \subseteq up_R^* \cup dn_R^*$

(ii) $dn_R \circ Q \subseteq Q \circ dn_R$

The inclusion (i) is justified as follows.

$$up_R^* \circ dn_R^* \subseteq up_R^* \cup dn_R^*$$
$\Leftarrow$ μ-induction
$$dn_R^* \cup up_R \circ (up_R^* \cup dn_R^*) \subseteq up_R^* \cup dn_R^*$$
$\equiv$ μ-computation; distribution
$$dn_R^* \cup up_R \circ up_R^* \cup up_R \circ dn_R \circ dn_R^* \subseteq up_R^* \cup dn_R^*$$
$\Leftarrow$ $up_R \circ dn_R \subseteq id;\ up_R \circ up_R^* \subseteq up_R^*$

true

The inclusion (ii) is shown as follows.

$$dn_R \circ Q \subseteq Q \circ dn_R$$
$\equiv$ μ-computation; distributivity; $up_L \circ dn_R = \emptyset$
$$dn_R \circ Q \subseteq dn_R \circ Q \circ up_R \circ dn_R$$
$\equiv$ commutativity of coreflexives
$$dn_R \circ Q \subseteq dn_R \circ up_R \circ dn_R \circ Q$$
$\equiv$ $dn_R \circ up_R \circ dn_R = dn_R$

true

Now we show the equation (4.1). From lemma (i) we derive as follows.

$$Q \circ dn_R^* \circ dn_L \subseteq dn_R^* \circ dn_L \circ \alpha?$$
$\Leftarrow$ $(R)[S]^* \subseteq S^{\smile *} \circ R \circ S^*$ for all R, S
$$dn_R^* \circ dn_L \circ \alpha? \circ up_L \circ up_R^* \circ dn_R^* \circ dn_L \subseteq dn_R^* \circ dn_L \circ \alpha?$$
$\Leftarrow$ monotonicity
$$up_L \circ up_R^* \circ dn_R^* \circ dn_L \subseteq id$$
$\Leftarrow$ lemma (i)
$$up_L \circ (up_R^* \cup dn_R^*) \circ dn_L \subseteq id$$
$\equiv$ μ-computation; distributivity
$$up_L \circ dn_L \cup$$
$$up_L \circ up_R^* \circ up_R \circ dn_L \cup up_L \circ dn_R \circ dn_R^* \circ dn_L \subseteq id$$
$\Leftarrow$ $up_L \circ dn_L \subseteq id;\ up_R \circ dn_L = \emptyset;\ up_L \circ dn_R = \emptyset$

true

The converse inclusion is a consequence of lemma (ii).

$$dn_R^* \circ dn_L \circ \alpha? \subseteq Q \circ dn_R^* \circ dn_L$$
$\Leftarrow$ μ-induction
$$dn_L \circ \alpha? \cup dn_R \circ Q \circ dn_R^* \circ dn_L \subseteq Q \circ dn_R^* \circ dn_L$$
$\Leftarrow$ lemma (ii)
$$dn_L \circ \alpha? \cup Q \circ dn_R \circ dn_R^* \circ dn_L \subseteq Q \circ dn_R^* \circ dn_L$$
$\equiv$ $dn_R \circ dn_R^* \subseteq dn_R^*$
$$dn_L \circ \alpha? \subseteq Q \circ dn_R^* \circ dn_L$$
$\Leftarrow$ $dn_L \circ \alpha? \circ up_L \subseteq Q;\ id \subseteq dn_R^*$
$$dn_L \circ \alpha? \subseteq dn_L \circ \alpha? \circ up_L \circ dn_L$$
$\equiv$ commutativity of coreflexives; $dn_L \circ up_L \circ dn_L = dn_L$

true □

4.2 The descendant/ancestor symmetry

By analogy to the previous example, one might expect that two XPath expressions desc :: β[anc :: α] and desc-or-self :: α/desc :: β are equivalent, but not indeed. The former expression enumerates all the descendant β nodes which have an α node as its ancestor, but the latter enumerates only such β nodes whose α ancestor node is found as the current node or its descendant. They are nevertheless equated when they are prefixed by the absolute path '/', that is,

/desc :: β[anc :: α] and /desc-or-self :: α/desc :: β

are equivalent.

To establish this equivalence, we begin with a finer comparison between the relative path expressions. Let us show that

desc :: β[anc :: α] is equal to
(desc-or-self :: α/desc :: β) ∪ (self :: ∗[anc :: α]/desc :: β),

which amounts to showing the following equality of relations.

$$\begin{aligned} & dom\ (\alpha? \circ (up_L \circ up_R^*)^+) \circ \beta? \circ (dn_R^* \circ dn_L)^+ \\ = \ & \beta? \circ (dn_R^* \circ dn_L)^+ \circ \alpha? \circ (dn_R^* \circ dn_L)^* \cup \qquad (4.2) \\ & \beta? \circ (dn_R^* \circ dn_L)^+ \circ dom\ (\alpha? \circ (up_L \circ up_R^*)^+) \circ (\mathtt{Node}? \times id) \end{aligned}$$

Let us write C for $dn_R^* \circ dn_L$ and Q for $(dn_L \circ \alpha? \circ up_L)[up_L, up_R]^*$. Since $(up_L \circ up_R^*)^+ = up_L \circ (up_L \cup up_R)^*$, we have $dom\ (\alpha? \circ (up_L \circ up_R^*)^+) = Q$ by (SYMCLS-DOM) and hence, by the commutativity of coreflexives, monotonicity, and the fact that $dom\ dn_L \subseteq \mathtt{Node}? \times id$, it suffices to show the following inclusions.

$$C^+ \circ \alpha? \circ C^* \subseteq Q \circ C^+ \qquad (4.3)$$
$$C^+ \circ Q \subseteq Q \circ C^+ \qquad (4.4)$$
$$Q \circ C^+ \subseteq C^+ \circ \alpha? \circ C^* \cup C^+ \circ Q \qquad (4.5)$$

We first show a few subsidiary lemmas:

(i) $C \circ Q \subseteq Q \circ C$
(ii) $Q \circ C \subseteq C \circ (Q \cup \alpha?)$

Lemma (i) follows from the facts that $dn_L \circ Q \subseteq Q \circ dn_L$ and $dn_R \circ Q \subseteq Q \circ dn_R$. We will show $dn_L \circ Q \subseteq Q \circ dn_L$. (The other inclusion is similarly derived.)

$$\begin{aligned} & dn_L \circ Q \subseteq Q \circ dn_L \\ \equiv \quad & \mu\text{-computation; distributivity} \\ & dn_L \circ Q \subseteq dn_L \circ \alpha? \circ up_L \circ dn_L \cup dn_R \circ Q \circ up_R \circ dn_L \cup \\ & \qquad dn_L \circ Q \circ up_L \circ dn_L \\ \Leftarrow \quad & up_R \circ dn_L = \emptyset \\ & dn_L \circ Q \subseteq dn_L \circ Q \circ up_L \circ dn_L \\ \Leftarrow \quad & \text{commutativity of coreflexives; } dn_L = dn_L \circ up_L \circ dn_L \\ & \mathbf{true} \end{aligned}$$

We calculate as follows for lemma (ii).

$$\begin{aligned} & Q \circ C \subseteq C \circ (Q \cup \alpha?) \\ \Leftarrow \quad & Q \circ dn_R^* \subseteq dn_R^* \circ Q \ (\dagger) \\ & dn_R^* \circ Q \circ dn_L \subseteq C \circ (Q \cup \alpha?) \\ \Leftarrow \quad & \mu\text{-induction} \\ & Q \circ dn_L \cup dn_R \circ C \circ (Q \cup \alpha?) \subseteq C \circ (Q \cup \alpha?) \\ \equiv \quad & dn_R \circ C \subseteq C \\ & Q \circ dn_L \subseteq C \circ (Q \cup \alpha?) \\ \equiv \quad & \mu\text{-computation} \\ & dn_L \circ \alpha? \circ up_L \circ dn_L \cup \\ & \quad dn_L \circ Q \circ up_L \circ dn_L \cup dn_R \circ Q \circ up_R \circ dn_L \subseteq C \circ (Q \cup \alpha?) \\ \equiv \quad & up_R \circ dn_L = \emptyset;\ up_L \circ dn_L \subseteq id;\ dn_L \subseteq C \\ & \mathbf{true} \end{aligned}$$

In the above calculation, the fact (†) is derived as below.

$$\begin{aligned} & Q \circ dn_R^* \subseteq dn_R^* \circ Q \\ \Leftarrow \quad & \mu\text{-induction} \\ & Q \cup dn_R^* \circ Q \circ dn_R \subseteq dn_R^* \circ Q \\ \Leftarrow \quad & \mu\text{-computation; monotonicity} \\ & Q \circ dn_R \subseteq dn_R \circ Q \\ \equiv \quad & \mu\text{-computation; distributivity} \\ & dn_L \circ \alpha? \circ up_L \circ dn_R \cup \\ & \quad dn_L \circ Q \circ up_L \circ dn_R \cup dn_R \circ Q \circ up_R \circ dn_R \subseteq dn_R \circ Q \\ \Leftarrow \quad & up_L \circ dn_R = \emptyset;\ up_R \circ dn_R \subseteq id \\ & \mathbf{true} \end{aligned}$$

The inclusion (4.3) is calculated as follows.

$$\begin{aligned} & C^+ \circ \alpha? \circ C^* \subseteq Q \circ C^+ \\ \equiv \quad & C^+ = C^* \circ C \\ & C^* \circ C \circ \alpha? \circ C^* \subseteq Q \circ C^+ \\ \Leftarrow \quad & \mu\text{-induction} \\ & C \circ \alpha? \circ C^* \cup C \circ Q \circ C^+ \subseteq Q \circ C^+ \\ \Leftarrow \quad & C \circ Q \subseteq Q \circ C \\ & C \circ \alpha? \circ C^* \subseteq Q \circ C^+ \\ \Leftarrow \quad & \mu\text{-induction} \\ & C \circ \alpha? \cup Q \circ C^+ \circ C \subseteq Q \circ C^+ \\ \Leftarrow \quad & C^+ \circ C \subseteq C^+ \\ & dn_R^* \circ dn_L \circ \alpha? \subseteq Q \circ C \\ \Leftarrow \quad & \mu\text{-induction} \\ & dn_L \circ \alpha? \cup dn_R \circ Q \circ C \subseteq Q \circ C \\ \Leftarrow \quad & dn_R \circ Q \subseteq Q \circ dn_R \\ & dn_L \circ \alpha? \cup Q \circ dn_R \circ C \subseteq Q \circ C \\ \Leftarrow \quad & dn_R \circ C \subseteq C;\ dn_L \circ \alpha? \circ up_L \circ dn_L \subseteq Q \circ C \\ & dn_L \circ \alpha? \subseteq dn_L \circ \alpha? \circ up_L \circ dn_L \\ \Leftarrow \quad & \text{commutativity of coreflexives; } dn_L = dn_L \circ up_L \circ dn_L \\ & \mathbf{true} \end{aligned}$$

The inclusion (4.4) follows from lemma (i). The inclusion (4.5) is derived as follows.

$$\begin{aligned} & Q \circ C^+ \subseteq C^+ \circ \alpha? \circ C^* \cup C^+ \circ Q \\ \Leftarrow \quad & \mu\text{-induction; distributivity} \\ & Q \circ C \cup C^+ \circ \alpha? \circ C^* \circ C \cup C^+ \circ Q \circ C \subseteq C^+ \circ \alpha? \circ C^* \cup C^+ \circ Q \\ \equiv \quad & C^* \circ C \subseteq C^* \\ & Q \circ C \cup C^+ \circ Q \circ C \subseteq C^+ \circ \alpha? \circ C^* \cup C^+ \circ Q \\ \equiv \quad & \text{lemma (ii)} \\ & C^+ \circ Q \circ C \subseteq C^+ \circ \alpha? \circ C^* \cup C^+ \circ Q \\ \Leftarrow \quad & \mu\text{-induction} \\ & C \circ Q \circ C \cup C \circ (C^+ \circ \alpha? \circ C^* \cup C^+ \circ Q) \subseteq C^+ \circ \alpha? \circ C^* \cup C^+ \circ Q \\ \Leftarrow \quad & \text{distributivity; } C \circ C^+ \subseteq C^+\text{; lemma (ii)} \\ & C \circ C \circ (Q \cup \alpha?) \subseteq C^+ \circ \alpha? \circ C^* \cup C^+ \circ Q \\ \Leftarrow \quad & \text{distributivity; } C \circ C \subseteq C^+;\ id \subseteq C^* \\ & \mathbf{true} \end{aligned}$$

□

As a corollary to the equation (4.2), we establish the equality between the absolute path expressions:

$$I_p[\![/\text{desc} :: \beta[\text{anc} :: \alpha]]\!] \ = \ I_p[\![/\text{desc-or-self} :: \alpha/\text{desc} :: \beta]\!].$$

This follows from the fact that the absolute path /self :: ∗[anc :: α]/desc :: β is interpreted as the empty relation, which is a consequence of the following lemma:

$$(dn_L \circ \alpha? \circ up_L)[up_L, up_R]^* \circ (id \times \mathtt{Top}?) \subseteq \emptyset. \qquad (4.6)$$

This lemma is shown by the following calculuation.

$$
\begin{array}{ll}
 & (dn_L \circ \alpha? \circ up_L)[up_L, up_R]^* \circ (id \times \mathtt{Top?}) \subseteq \emptyset \\
\equiv & \quad \mu\text{-computation; distributivity} \\
 & dn_L \circ \alpha? \circ up_L \circ (id \times \mathtt{Top?}) \cup \\
 & dn_L \circ (dn_L \circ \alpha? \circ up_L)[up_L, up_R]^* \circ up_L \circ (id \times \mathtt{Top?}) \cup \\
 & dn_R \circ (dn_L \circ \alpha? \circ up_L)[up_L, up_R]^* \circ up_R \circ (id \times \mathtt{Top?}) \ \subseteq \emptyset \\
\equiv & \quad up_L \circ (id \times \mathtt{Top?}) = \emptyset;\ up_R \circ (id \times \mathtt{Top?}) = \emptyset \\
 & \mathbf{true} \qquad \square
\end{array}
$$

5. Calculating with negative predicates

So far we have dealt with positive fragments of XPath expressions only. It is not difficult to give a relational specification for negative predicates. We interpret negative predicates by

$$I_q[\![\text{not } q]\!] = \neg I_q[\![q]\!],$$

where $\neg$ is the complement on coreflexive relations R, which is defined by $\neg R = id - R$.

Some basic properties of negation is listed below. (As usual, $\neg$ binds tighter than the others.)

For any coreflexive relations R and S, we have:

$\neg R$ is coreflexive	(NEG-COREFL)
$\neg R$ is anti-monotonic in R	(NEG-ANTIMONO)
$\neg\emptyset = id$ and $\neg id = \emptyset$	(NEG-CMPL)
$\neg\neg R = R$	(NEG-DBLNEG)
$\neg(R \cup S) = \neg R \cap \neg S$ $\neg(R \cap S) = \neg R \cup \neg S$	(NEG-DEMORGAN)
$R \cup \neg R = id$ and $R \cap \neg R = \emptyset$	(NEG-EXMDL)

These properties allow us to treat negations as those in propositional logic formulas.

5.1 Calculating relational negations

For effective calculation of negative predicates, we make use of the following laws for simplifying expressions of the form $\neg\, dom\ R$.

$$\neg dom\ (R \circ S) = \neg dom\ S \cup S^{\smile} \circ (\neg dom\ R) \circ S, \quad \text{(NEG-DOM)}$$
if S is simple and injective.

$$\neg dom\ (R \circ S) = \neg S \cup (\neg dom\ R) \circ S, \quad \text{(NEG-DOMCOREFL)}$$
if S is coreflexive.

Let us show (NEG-DOM). (The law (NEG-DOMCOREFL) immediately follows from (NEG-DOM).)

For one inclusion, we derive:

$$
\begin{array}{ll}
 & \neg dom\ (R \circ S) \subseteq \neg dom\ S \cup S^{\smile} \circ (\neg dom\ R) \circ S \\
\equiv & \quad \text{(GC-DIFF), twice} \\
 & \neg\neg dom\ S \subseteq dom\ (R \circ S) \cup S^{\smile} \circ (\neg dom\ R) \circ S \\
\equiv & \quad \text{(NEG-DBLNEG); (DOM-COMP)} \\
 & S^{\smile} \circ S \subseteq S^{\smile} \circ (dom\ R) \circ S \cup S^{\smile} \circ (\neg dom\ R) \circ S \\
\equiv & \quad \text{monotonicity; distributivity} \\
 & id \subseteq dom\ R \cup \neg dom\ R \\
\equiv & \quad \text{(NEG-EXMDL)} \\
 & \mathbf{true}
\end{array}
$$

For the other inclusion, we show both $\neg dom\ S \subseteq \neg dom\ (R \circ S)$ and $S^{\smile} \circ (\neg dom\ R) \circ S \subseteq \neg dom\ (R \circ S)$.

$$
\begin{array}{ll}
 & \neg dom\ S \subseteq \neg dom\ (R \circ S) \\
\Leftarrow & \quad \text{anti-monotonicity} \\
 & dom\ (R \circ S) \subseteq dom\ S \\
\Leftarrow & \quad S \text{ is injective} \\
 & S^{\smile} \circ dom\ R \circ S \subseteq S^{\smile} \circ S \\
\Leftarrow & \quad dom\ R \text{ is coreflexive} \\
 & \mathbf{true}
\end{array}
$$

$$
\begin{array}{ll}
 & S^{\smile} \circ (\neg dom\ R) \circ S \subseteq \neg dom\ (R \circ S) \\
\equiv & \quad \text{(NEG-DBLNEG); (GC-DIFF); (DOM-COMP)} \\
 & id \subseteq \neg(S^{\smile} \circ (\neg dom\ R) \circ S) \cup \neg(S^{\smile} \circ dom\ R \circ S) \\
\equiv & \quad \text{anti-monotonicity; de Morgan's law; } \neg id = \emptyset \\
 & S^{\smile} \circ (\neg dom\ R) \circ S \cap S^{\smile} \circ dom\ R \circ S \subseteq \emptyset \\
\Leftarrow & \quad \text{modular law, twice} \\
 & S^{\smile} \circ (\neg dom\ R \cap S \circ S^{\smile} \circ dom\ R \circ S \circ S^{\smile}) \circ S \subseteq \emptyset \\
\Leftarrow & \quad S \text{ is simple} \\
 & S^{\smile} \circ (\neg dom\ R \cap dom\ R) \circ S \subseteq \emptyset \\
\equiv & \quad \text{(NEG-EXMDL)} \\
 & \mathbf{true} \qquad \square
\end{array}
$$

Example 3 An XPath expression with a negative predicate is interpreted as follows.

$$
\begin{array}{l}
I_e[\![\text{desc} :: \beta[\text{not anc} :: \alpha]]\!] \\
\quad = \neg(dn_L \circ \alpha? \circ up_L)[up_L, up_R]^* \circ \beta? \circ (dn_R^* \circ dn_L)^+
\end{array}
$$

5.2 The descendant/ancestor symmetry with negation

As an example of calculation with negative predicates, we try to find a relational tree navigation that is equal to the following absolute XPath expression:

$$/\text{desc} :: \beta[\text{not anc} :: \alpha].$$

Similar to the calculation conducted in Section 4.2, we first show another equation corresponding to the relative path.

$$
\begin{array}{l}
\neg(dn_L \circ \alpha? \circ up_L)[up_L, up_R]^* \circ \beta? \circ (dn_R^* \circ dn_L)^+ \\
\quad = \beta? \circ (dn_R^* \circ dn_L \circ \neg\alpha?)^+ \circ \neg(dn_L \circ \alpha? \circ up_L)[up_L, up_R]^*
\end{array}
\tag{5.1}
$$

Let us write C for $dn_R^* \circ dn_L$ and Q for $(dn_L \circ \alpha? \circ up_L)[up_L, up_R]^*$. It is easy to show that the equations $C^+ = (dn_R \cup dn_L)^* \circ dn_L$ and $(C \circ \neg\alpha?)^+ = (dn_R \cup dn_L \circ \neg\alpha?)^* \circ dn_L \circ \neg\alpha?$ hold.

By the commutativity of coreflexives, it suffices to show the equality $(C \circ \neg\alpha?)^+ \circ \neg Q = \neg Q \circ C^+$.

We first show the following lemmas.

(i) $\neg Q \circ dn_R = dn_R \circ \neg Q$

(ii) $\neg Q \circ dn_L = dn_L \circ \neg\alpha? \circ \neg Q$

(iii) $\neg Q \circ C = C \circ \neg\alpha? \circ \neg Q$

By applying μ-computation, de Morgan's law, and (NEG-DOM), we obtain the equation $\neg Q = R_1 \cap R_2 \cap R_3$ where

$$
\begin{array}{l}
R_1 = \neg(dom\ up_L) \cup dn_L \circ \neg\alpha? \circ up_L, \\
R_2 = \neg(dom\ up_L) \cup dn_L \circ \neg Q \circ up_L, \text{ and} \\
R_3 = \neg(dom\ up_R) \cup dn_R \circ \neg Q \circ up_R.
\end{array}
$$

We will prove (ii). (Lemma (i) can be shown in a similar way.) Since $dn_L \circ \neg\alpha? \circ up_L \circ dn_L = dn_L \circ \neg\alpha?$ because of coreflexivity and $dn_L = dn_L \circ up_L \circ dn_L$, we have $R_1 \circ dn_L = \neg(dom\ up_L) \circ dn_L \cup dn_L \circ \neg\alpha?$. Further we calculate by the laws (DOM-DOMAIN), (COREFL-COMP), and (NEG-EXMDL):

$$\begin{aligned}\neg(dom\ up_L) \circ dn_L &= \neg(dom\ up_L) \circ (dom\ up_L) \circ dn_L \\ &= (\neg(dom\ up_L) \cap (dom\ up_L)) \circ dn_L = \emptyset.\end{aligned}$$

Thus we have $R_1 \circ dn_L = dn_L \circ \neg\alpha?$. Similarly, we can derive $R_2 \circ dn_L = dn_L \circ \neg Q$. Further we have $R_3 \circ dn_L = dn_L$ because $up_R \circ dn_L = \emptyset$ and also $dn_L \subseteq \neg(dom\ up_R) \circ dn_L$, as shown below.

$$\begin{aligned}& dn_L \subseteq \neg(dom\ up_R) \circ dn_L \\ \Leftarrow\quad & \text{(DOM-DOMAIN); monotonicity} \\ & dom\ up_L \subseteq \neg(dom\ up_R) \\ \equiv\quad & \text{(NEG-DBLNEG); (GC-DIFF); de Morgan} \\ & id \subseteq \neg(dom\ up_L \cap dom\ up_R) \\ \equiv\quad & dom\ up_L \cap dom\ up_R = dn_L \circ up_L \circ dn_R \circ up_R = \emptyset \\ & \mathbf{true}\end{aligned}$$

Therefore (ii) can be shown as follows.

$$\begin{aligned}\neg Q \circ dn_L = &\ (R_1 \cap R_2 \cap R_3) \circ dn_L = R_1 \circ R_2 \circ R_3 \circ dn_L \\ = &\ R_1 \circ R_2 \circ dn_L = R_1 \circ dn_L \circ \neg Q = dn_L \circ \neg\alpha? \circ \neg Q\end{aligned}$$

Lemma (iii) is a consequence of (i) and (ii), as shown below. (The other inclusion is proved in a similar way using μ-induction.)

$$\begin{aligned}& \neg Q \circ C \subseteq C \circ \neg\alpha? \circ \neg Q \\ \equiv\quad & \text{(CLOSURE-UEP)} \\ & \neg Q \circ C \subseteq \nu X.(dn_L \circ \neg\alpha? \circ \neg Q \cup dn_R \circ X) \\ \Leftarrow\quad & \nu\text{-induction} \\ & \neg Q \circ C \subseteq dn_L \circ \neg\alpha? \circ \neg Q \cup dn_R \circ \neg Q \circ C \\ \equiv\quad & \text{(i) and (ii)} \\ & \neg Q \circ C \subseteq \neg Q \circ dn_L \cup \neg Q \circ dn_R \circ C \\ \equiv\quad & \text{distributivity; } \mu\text{-computation} \\ & \mathbf{true}\end{aligned}$$

Now we show the inclusion $(C \circ \neg\alpha?)^+ \circ \neg Q \subseteq \neg Q \circ C^+$.

$$\begin{aligned}& (C \circ \neg\alpha?)^+ \circ \neg Q \subseteq \neg Q \circ C^+ \\ \Leftarrow\quad & \mu\text{-induction} \\ & C \circ \neg\alpha? \circ \neg Q \cup C \circ \neg\alpha? \circ \neg Q \circ C^+ \subseteq \neg Q \circ C^+ \\ \equiv\quad & \text{(iii)} \\ & \neg Q \circ C \cup \neg Q \circ C \circ C^+ \subseteq \neg Q \circ C^+ \\ \equiv\quad & \mu\text{-computation} \\ & \mathbf{true}\end{aligned}$$

For the other inclusion, we calculate:

$$\begin{aligned}& \neg Q \circ C^+ \subseteq (C \circ \neg\alpha?)^+ \circ \neg Q \\ \Leftarrow\quad & \mu\text{-induction} \\ & \neg Q \circ C \cup (C \circ \neg\alpha?)^+ \circ \neg Q \circ C \subseteq (C \circ \neg\alpha?)^+ \circ \neg Q \\ \equiv\quad & \text{(iii)} \\ & C \circ \neg\alpha? \circ \neg Q \cup (C \circ \neg\alpha?)^+ \circ C \circ \neg\alpha? \circ \neg Q \subseteq (C \circ \neg\alpha?)^+ \circ \neg Q \\ \equiv\quad & \text{monotonicity; } \mu\text{-computation} \\ & \mathbf{true} \qquad \square\end{aligned}$$

The absolute path specification we consider is then equated to a relational specification as follows.

$$\begin{aligned}& I_e[\![/\text{desc} :: \beta[\text{not anc} :: \alpha]]\!] \\ = &\ \beta? \circ (dn_R^* \circ dn_L \circ \neg\alpha?)^+ \circ (id \times \mathtt{Top}?) \circ (up_L \cup up_R)^*\end{aligned} \tag{5.2}$$

To establish this equation via (5.1), it is sufficient to show the equation:

$$\neg(dn_L \circ \alpha? \circ up_L)[up_L, up_R]^* \circ (id \times \mathtt{Top}?) = id \times \mathtt{Top}?.$$

Writing Q for $(dn_L \circ \alpha? \circ up_L)[up_L, up_R]^*$, we derive this equation as follows.

$$\begin{aligned}& \neg Q \circ (id \times \mathtt{Top}?) = id \times \mathtt{Top}? \\ \equiv\quad & \text{(COREFL-COMP); monotonicity} \\ & id \times \mathtt{Top}? \subseteq \neg Q \\ \equiv\quad & \text{(NEG-DBLNEG); (GC-DIFF); de Morgan} \\ & id \subseteq \neg(Q \cap (id \times \mathtt{Top}?)) \\ \equiv\quad & \text{anti-monotonicity; } \neg id = \emptyset\text{; (COREFL-COMP)} \\ & Q \circ (id \times \mathtt{Top}?) \subseteq \emptyset \\ \equiv\quad & (4.6) \\ & \mathbf{true} \qquad \square\end{aligned}$$

Here we notice that the resulting relational specification in (5.2) is implementable as a functional program (see the next section) but not expressible by an XPath expression.

6. Translating relations to programs

This section briefly describes how relational specifications of tree navigation can be translated into functional programs. As most of the translation process comprises of standard techniques such as power transpose (see e.g. [4]), we elaborate mainly on what are specific to the present work.

To fill the gap between relational specifications and functional programs, we need a few rearrangements on the former. First, we rewrite the relation into an equivalent one that has no nested closures and more explicit conditional choices. For this, we rewrite, e.g., the relation $(dn_R^* \circ dn_L \circ \neg\alpha?)^+$ derived in Section 5 into $(dn_R \cup dn_L \circ \neg\alpha?)^* \circ\ dn_L \circ \neg\alpha?$. Second, assuming coreflexivity of R and injectivity of S_1, ..., S_n, we deal with every symmetric closure $(S_1, \cdots, S_n)[R]^*$ or its negation, which is derived from an XML predicate, according to (SYMCLS-RCOND). For example, the XPath negative predicate [not anc :: α] in Section 5 is regarded as (the negation of) $\Pi \circ dn_L \circ \alpha? \circ up_L \circ (up_L \cup up_R)^*$, which is a right condition derived from $(dn_L \circ \alpha? \circ up_L)[up_L, up_R]^*$. This use of right conditions in place of coreflexives is intended for efficiency, where (SYMCLS-RCOND) allows us to skip almost half of the tree navigation in the symmetric closure.

Once prepared as above, we proceed the translation as below. We treat every coreflexive R as its right condition $\Pi \circ R$ and translate it to a boolean-valued function whose output is true if and only if something is related by R with the input. The relational meets, joins, and negation should be interpreted as corresponding operations on boolean values. Relational compositions of coreflexives should be treated as conjunction, as indicated by (COREFL-COMP). Every atomic predicate, which is derived from an XPath step path, should be replaced with a suitable node test. For the other (not coreflexive) relations, we interpret them to its power transpose [4]: Given a relation R over zippers, the power transpose of R, written ΛR, is a function that takes a zipper and returns a set of zippers related by R. Adopting lists as the conventional representation for sets, we can implement ΛR by a function that returns a list of zippers that are related by R to the input zipper and we also replace each relational union with the list append.

```
let rec root z =
  match z with
    (_,Top) -> z
  | (_,L _) -> root (upL z)
  | (_,R _) -> root (upR z)

let node a =
  function (Node(l,i,r),p) -> a=i | _ -> false

let rec pred_anc a z =
  match z with
    (Leaf,_) | (_, Top) -> false
  | (Node _, L _) ->
      let u = upL z in node a u || pred_anc a u
  | (Node _, R _) -> pred_anc a (upR z)

let desc b z =
  let rec dn z =
    match z with
      (Node _,_) ->
        List.append (if node b z then [z] else [])
          (List.append (dn (dnL z)) (dn (dnR z)))
    | (Leaf,_) -> []
  in match z with
    (Node _,_) -> dn (dnL z)
  | _ -> []

let abs_desc_npred_anc a b z =
  let npred_anc a z = not (pred_anc a z)
  in List.filter (npred_anc a) (desc b (root z))

let nadesc a z =
  let rec nadcsf z =
  match z with
    (Leaf,_) -> [z]
  | (Node _, _) ->
      List.append [z]
        (List.append (nadcsf (dnR z))
          (if not (node a z)
           then nadcsf (dnL z) else []))
  in match z with
       (Leaf,_) -> []
     | (Node _,_) ->
         if not (node a z)
         then nadcsf (dnL z) else []

let abs_b_nadesc a b z =
  List.filter (node b) (nadesc a (root z))
```

Figure 5. A translation result

The translation procedures described above are mostly mechanical, but it would be difficult to fully automate the whole tranlation process because human insights are required in some translation steps, e.g., closure unnesting.

Figure 5 gives the result of translation of the relations that we examined in Section 5, namely, abs_desc_npred_anc for the relation corresponding to the negatively predicated absolute path expression $/\mathrm{desc} :: \beta[\mathrm{not\ anc} :: \alpha]$ and abs_b_nadesc for its equivalent relation $\beta? \circ (dn_R^* \circ dn_L \circ \neg\alpha?)^+ \circ (id \times \mathtt{Top}?) \circ (up_L \cup up_R)^*$ derived by calculation. Although the given program has chances of further optimization by other transformation techniques such as fusion [4], we leave them unexploited for readability.

7. Conclusion

We have proposed a theory for reasoning about equalities of relationally specified tree navigation. A small number of relational elements, tree navigation primitives and a few operations such as the symmetric closure, suffice for reasoning about both positive and negative XPath expressions and even those navigation beyond XPath expressibility.

There are several directions for future research based on the present work. First, we have not considered data updates in zippers at all. Developing primitive operations for data updates and a set of laws for reasoning would make our result more widely applicable. The result on negative predicates in Section 5 suggests that the XPath language is missing some symmetry induced from negative predicates. Identifying an extended path language that is closed under this symmetry would provide a fruitful merit with the language design. Although the present paper concerns only binary trees, we believe that the present work can be generalized to more varieties of data structures, based on the general zipper-like data structures as investigated in [1, 11]. Developing a theory for navigation in such general zipper-like data structures would also be an interesting topic to pursue.

Acknowledgments

The second author was partly supported by KAKENHI 20500001.

References

[1] M. Abbott, T. Altenkirch, C. McBride, and N. Ghani. δ for data: Differentiating data structures. *Fundamenta Informaticae*, 65(1–2): 1–28, 2004.

[2] R. Backhouse, R. Crole, and J. Gibbons, editors. *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction: International Summer School and Workshop*, volume 2297 of *LNCS*. Springer, 2002.

[3] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. *XML Path Language (XPath) 2.0, W3C Recommendation*, 2007. http://www.w3.org/TR/xpath20/.

[4] R. Bird and O. de Moor. *Algebra of Programming*, volume 100 of *International Series in Computer Science*. Prentice Hall, 1997.

[5] D. Che, K. Aberer, and M. T. Özsu. Query optimization in XML structured-document databases. *The VLDB Journal*, 15(3):263–289, 2006.

[6] A. Cunha and J. Visser. Transformation of structure-shy programs: Applied to XPath queries and strategic functions. In *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation*, pages 11–20. ACM, 2007.

[7] H. Doornbos, R. Backhouse, and J. van der Woude. A calculational approach to mathematical induction. *Theoretical Computer Science*, 179(1–2):103–135, 1997.

[8] P. Genevès, N. Layaïda, and A. Schmitt. Efficient static analysis of XML paths and types. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 342–351. ACM Press, 2007.

[9] G. Huet. The zipper. *Journal of Functional Programming*, 7(5):549–554, 1997.

[10] Mathematics of Program Construction Group. Fixed-point calculus. *Information Processing Letters*, 53(3):131–136, 1995.

[11] C. McBride. The derivative of a regular type is its type of one-hole contexts. Available electronically http://www.cs.nott.ac.uk/~ctm/diff.ps.gz., 2001.

[12] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. Technical Report PMS-FB-2001-17, University of Munich, 2001.

[13] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. In *XML-Based Data Management and Multimedia Engineering — EDBT 2002 Workshops*, volume 2490 of *LNCS*, pages 109–127. Springer, 2002.

[14] P. Wadler. A formal semantics of patterns in XSLT and XPATH. *Markup Languages: Theory and Practice*, 2(2):183–202, 2000.