Theory Comput Syst (2009) 44: 561–589
DOI 10.1007/s00224-008-9151-9

Axiomatizing the Logical Core of XPath 2.0

Balder ten Cate · Maarten Marx

Published online: 25 October 2008

Abstract The first aim of this paper is to present the logical core of XPath 2.0: a logically clean, decidable fragment, which includes most navigational features of XPath 2.0 (complex counting conditions and data joins are not supported, as they lead to undecidability). The second aim is to provide a list of equations completely axiomatizing query equivalence in this language (i.e., all other query equivalences can be derived from these).

Keywords XML · XPath · Axiomatization · Query rewriting

1 Introduction

The introduction of *Core XPath*, the navigational core of XPath 1.0 [11], has been a very fruitful move. It has given rise to many new results enhancing our understanding of XPath. The full language of XPath 1.0 is too rich for a rigorous logical analysis, and Core XPath is a concise and well defined sublanguage, for which a detailed analysis *is* feasible (e.g., concerning its expressive power and complexity). Many of these results were established by relating Core XPath to other, more familiar logical languages on trees such as *first-order logic* or *monadic second-order logic*, and applying known results and techniques for the latter.

A preliminary version of this paper was published in the proceedings of ICDT 2007.
We would like to thank Jan Hidders for pointing us to the interesting question of stronger forms of completeness (cf. Sect. 4.3). We would also like to thank the anonymous reviewers for their extensive comments, which have greatly improved the quality of the paper.
The first author is supported by NWO grant 639.021.508.

B. ten Cate (✉) · M. Marx
ISLA–Informatics Institute, Universiteit van Amsterdam, Kruislaan 403, 1098SJ Amsterdam, The Netherlands
e-mail: balder.tencate@uva.nl

M. Marx
e-mail: marx@science.uva.nl

XPath 2.0, the successor of XPath 1.0, has received less attention in the theoretical literature. In particular, no concrete proposal for its logical core has been made. One feature of XPath 2.0 is that it is expressively complete for first-order logic (in fact, it was designed to be). This does not mean that there are no interesting and challenging open problems for this language! In this paper we address one interesting problem, concerning query equivalence.

We identify the logical core of XPath 2.0, which we call *Core XPath 2.0*, and we present a complete axiomatization of query equivalence for this language, as well as for its variable free fragment, in the form of a finite list of remarkably simple algebraic equations. Our results might serve as a step towards improved *query optimization* methods for XPath 2.0 and XQuery (note that XQuery contains XPath 2.0 as a sublanguage).

Our axiomatization builds on a number of other results, including (i) a known complete axiomatization of Tarski's algebra of binary relations [20, 24], which can be seen as a sublanguage of Core XPath 2.0, and (ii) an axiomatization of the first-order theory of finite trees, cf. [2, 9].

To summarize, the main contributions are:

1. the definition of the logical core of XPath 2.0, together with an analysis of the redundancies in the language
2. a finite list of equations axiomatizing query equivalence, both for Core XPath 2.0 and for its variable free fragment.

Related Work In [4], an axiomatization was already given for a fragment of XPath 1.0 without node tests and with only the downward axes. Reference [22] contains axiomatizations of Core XPath and the child-only and descendant-only fragments of Core XPath. It is worth noting that, in *relational* database theory, complete axiomatizations of query equivalence are rather scarce. SQL and Datalog both have undecidable query equivalence problems, and hence query equivalence is not recursively axiomatizable for these languages (on finite structures). Entailment relations between different types of constraints *have* been successfully axiomatized [1].

From a more technical point of view, the *axiomatic-completeness-via-expressive-completeness* approach we follow was also employed in [25, 27] in the context of temporal logics over well-ordered flows of time, as was pointed out to us by Tadeusz Litak (personal communication) while we were preparing the final version of this paper.

Organization of the Paper Section 2 defines Core XPath 2.0 and establishes some simple equivalences. We also show undecidability of query equivalence for some modest extensions of Core XPath 2.0. In Sect. 3, we discuss some important connections between Core XPath 2.0, first-order logic, and Tarski's relation algebra, which are used in our proofs later on. Section 4 contains the axiomatizations for Core XPath 2.0 and its variable free fragment. We conclude in Sect. 5. The appendix contains a quick completeness proof for an axiomatization of the first-order theory of finite trees.

2 A Decidable Logical Core of XPath 2.0

In this section, we define the syntax and semantics of the logical core of XPath 2.0, and we discuss some basic properties of this language, as well as how it relates to other languages (viz. *first-order logic* and *relation algebra*).

2.1 Design Choices

The following two criteria guided our choice of XPath 2.0 operators to be included in the navigational core: (i) expressions should manipulate *sets of nodes*, just as in Core XPath. This leads to a simple set theoretic semantics; (ii) the query equivalence problem should be decidable. Undecidability would imply non-axiomatizability, because the models we are concerned with are finite.

These criteria have the following repercussions:

1. In Core XPath 2.0, all expressions manipulate sets of nodes. More precisely, the meaning of each path expression is a function that, given a node (the "*context node*") returns a set of nodes (the "*answer set*"). In contrast, in XPath 2.0, path expressions return *sequences* of elements, of various types. In particular, the `for` construct returns unsorted sequences, possibly containing duplicates. In our logic, we treat `for $i in` R `return` S as equivalent to `(for $i in` R `return` S`)/.`, which always returns a document-order sorted list of nodes without duplicates.
2. In order to keep the logic decidable, we excluded the positional and aggregate functions `position()`, `last()` and `count()` and value comparison operators. Section 2.4 presents undecidability results in the presence of these functions.

The crucial differences with Core XPath are that besides union, also Boolean intersection and complementation can be applied to path expressions, and that the language contains variables, node comparison tests, and the `for`-loop.

2.2 Syntax and Semantics of Core XPath 2.0

We are ready to define the *Navigational Core of XPath 2.0*. The grammar of Core XPath 2.0 is given in Fig. 1. Just like in Core XPath, it has productions for path expressions PathExpr and filter expressions TestExpr. In addition, there is the node comparison expression CompTest. By the *variable free fragment* of Core XPath 2.0 we will mean the fragment without variables, `for`, and node comparison tests.

The semantics is provided in Fig. 2, where 'dom' stands for the set of all nodes of the tree. Expressions are evaluated on finite unranked node-labeled and sibling-ordered trees as usual. If no confusion arises, we just call them trees.

Trees are defined as follows: A tree domain N is a finite set of finite sequences of natural numbers closed under taking initial segments, and for any sequence s, if $s \cdot k \in N$, then either $k = 0$ or $s \cdot k - 1 \in N$. A node labeled, sibling ordered tree consists of a tree domain and a function labeling each node with a set of primitive symbols from some alphabet. Node labeled, sibling ordered trees can be queried in a first order language in an appropriate signature. The signature consists of two binary relation symbols, $<$ and $\prec$, denoting the descendant and the following sibling

```
Axis      := self | child | parent | descendant | ancestor
               | following_sibling | preceding_sibling
NameTest  := QName | *
Step      := Axis::NameTest
NodeRef   := . | $i

PathExpr  := Step | NodeRef | ()
               | PathExpr/PathExpr
               | PathExpr union PathExpr
               | PathExpr intersect PathExpr
               | PathExpr except PathExpr
               | PathExpr[TestExpr]
               | for $i in PathExpr return PathExpr

TestExpr  := PathExpr | CompTest | not TestExpr
               | TestExpr and TestExpr | TestExpr or TestExpr

CompTest  := NodeRef is NodeRef
```

Fig. 1 Syntax of Core XPath 2.0

$$
\begin{array}{lcl}
[\![\mathsf{Axis}::\mathsf{NameTest}]\!]^g & = & \{(x,y)\in \mathrm{dom}^2 \mid x(\mathsf{Axis})y \text{ and } y \text{ satisfies } \mathsf{NameTest}\}\\
[\![.]\!]^g & = & [\![\mathtt{self} :: *]\!]^g\\
[\![\mathtt{\$i}]\!]^g & = & \{(x,y)\in \mathrm{dom}^2 \mid g(\$i)=y\}\\
[\![()]\!]^g & = & \emptyset\\
[\![R/S]\!]^g & = & [\![R]\!]^g \circ [\![S]\!]^g\\
[\![R \;\mathtt{union}\; S]\!]^g & = & [\![R]\!]^g \cup [\![S]\!]^g\\
[\![R \;\mathtt{intersect}\; S]\!]^g & = & [\![R]\!]^g \cap [\![S]\!]^g\\
[\![R \;\mathtt{except}\; S]\!]^g & = & [\![R]\!]^g \setminus [\![S]\!]^g\\
[\![R[T]]\!]^g & = & \{(x,y)\in [\![R]\!]^g \mid y \in [\![T]\!]^g_{\mathsf{Test}}\}\\
[\![\mathtt{for}\,\mathtt{\$i}\;\mathtt{in}\,R\,\mathtt{return}\,S]\!]^g & = & \{(x,y)\in \mathrm{dom}^2 \mid \exists z.\ (x,z)\in [\![R]\!]^g \text{ and } (x,y)\in [\![S]\!]^{g_z^{\$i}}\}\\
 & & \\
[\![\mathsf{PathExpr}]\!]^g_{\mathsf{Test}} & = & \{x\in \mathrm{dom} \mid \exists y.\ (x,y)\in [\![\mathsf{PathExpr}]\!]^g\}\\
[\![\mathtt{not}\; T]\!]^g_{\mathsf{Test}} & = & \mathrm{dom} \setminus [\![T]\!]^g_{\mathsf{Test}}\\
[\![T1 \;\mathtt{and}\; T2]\!]^g_{\mathsf{Test}} & = & [\![T1]\!]^g_{\mathsf{Test}} \cap [\![T2]\!]^g_{\mathsf{Test}}\\
[\![T1 \;\mathtt{or}\; T2]\!]^g_{\mathsf{Test}} & = & [\![T1]\!]^g_{\mathsf{Test}} \cup [\![T2]\!]^g_{\mathsf{Test}}\\
[\![a \;\mathtt{is}\; b]\!]^g_{\mathsf{Test}} & = & \{x\in \mathrm{dom} \mid [a]^{g,x} = [b]^{g,x}\}.
\end{array}
$$

Fig. 2 Semantics of the Navigational Core of XPath 2.0

relation, respectively; unary predicate symbols corresponding to the node labels; and equality. For $n, n' \in N$, $n < n'$ holds iff n is a proper prefix of n'; $n \prec n'$ holds iff $n = s \cdot k$, $n' = s \cdot l$ and $k < l$; $P(n)$ holds iff n is labeled by the symbol P.

Because of the variables in the language we need an assignment function g mapping variables to nodes. For g an assignment, $\$\mathtt{i}$ a variable and x a node, $g_x^{\$\mathtt{i}}$ denotes the assignment which is just like g except that $g_x^{\$\mathtt{i}}(\$\mathtt{i}) = x$. The value of a NodeRef expression a relative to an assignment g and a node x, denoted by $[a]^{g,x}$, is x if $a = .$, or $g(a)$ if a is a variable.

Given a tree and an assignment g, the meaning $[\![R]\!]^g$ of a PathExpr R is always a binary relation. Of course this is just another way of specifying a function from the set of nodes to the powerset of the set of nodes (the answer-set semantics).

For example, the path expression

```
for $i in descendant::author return $i/child::name
```

when evaluated in a node x, returns for each descendant node with tag `author`, all children with tag `name`. Hence, it defines the binary relation containing all pairs (x, y) with y a `name`-node whose parent is an `author`-node that is a descedant of x.

The meaning of a TestExpr expression is given by the function $[\![\cdot]\!]^g_{\mathsf{Test}}$, which always yields a set of nodes: $x \in [\![\mathsf{TestExpr}]\!]^g_{\mathsf{Test}}$ if and only if TestExpr evaluates to true at node x.

If an expression has no free variables (i.e., if all occurrences of variables are in the scope of corresponding `for`-operators), the assignment is irrelevant, and we will write $[\![R]\!]$ instead of $[\![R]\!]^g$. In particular, this applies to all expressions in the variable free fragment.

It is straightforward to check that the given semantics extends the semantics of Core XPath given in e.g. [11], and agrees with the official XPath 2.0 semantics as presented in e.g., [16], provided sequences are treated as nodesets.

Definition 1 Two path expressions R, S are said to be equivalent if for every tree, for every assignment g, $[\![R]\!]^g = [\![S]\!]^g$.

Remark We excluded the attribute axis because it only adds expressivity in the presence of data value comparisons. Because we focus on relative path equivalence, we also excluded the absolute path expression $/R$. This is term definable as $(\mathtt{ancestor}::*\ \mathtt{union}\ \mathtt{self}::*)[\mathtt{not}\ \mathtt{ancestor}::*]/R$.

2.3 Syntactic Sugar

XPath 2.0 contains a number of extra axes, operations and functions that, when restricted to the navigational fragment, are just syntactic sugar. They are listed in Fig. 3. As indicated, all these connectives can be term-defined in terms of the connectives of Fig. 1. With the exception of the `if-then-else` construct, all definitions are linear.

We will be making use of a slightly more compact notation in the following sections. We will use $\downarrow, \downarrow^+, \uparrow, \uparrow^+, \rightarrow, \rightarrow^+, \leftarrow, \leftarrow^+$ and . as shorthands for the respective axes `child::*`, `descendant::*`, `parent::*`, `ancestor::*`, `(following_sibling::* except following_sibling::*/following_sibling::*)`, `following_sibling::*`, `(preceding_sibling::*except preceding_sibling::*/preceding_sibling::*)`, `preceding_sibling::*`, and `self::*`. We will use $\top$ as shorthand for the universal relation $(\uparrow^+\ \mathtt{union}\ .)/(\downarrow^+\ \mathtt{union}\ .)$ and $\bot$ as shorthand for `()`. We will use $\ll$ to denote the document order (depth-first left-to-right) relation in a tree, as defined by $\downarrow^+\ \mathtt{union}\ (\uparrow^+\ \mathtt{union}\ .)/\rightarrow^+/(\downarrow^+\ \mathtt{union}\ .)$. Finally, we will use the function $(\cdot)^{\smile}$, which, when applied to a path expression, yields its converse:

Definable path expressions:

$$\begin{array}{ll}
\texttt{descendant\_or\_self} :: \texttt{NameTest} & \\
 & \equiv \texttt{descendant} :: \texttt{NameTest}\ \texttt{union}\ \texttt{self} :: \texttt{NameTest} \\
\texttt{ancestor\_or\_self} :: \texttt{NameTest} & \\
 & \equiv \texttt{ancestor} :: \texttt{NameTest}\ \texttt{union}\ \texttt{self} :: \texttt{NameTest} \\
\texttt{following} :: \texttt{NameTest} & \equiv (\texttt{ancestor} :: *\ \texttt{union}\ \texttt{self} :: *)/\texttt{following\_sibling} :: */ \\
 & \quad (\texttt{descendant} :: \texttt{NameTest}\ \texttt{union}\ \texttt{self} :: \texttt{NameTest}) \\
\texttt{preceding} :: \texttt{NameTest} & \equiv (\texttt{ancestor} :: *\ \texttt{union}\ \texttt{self} :: *)/\texttt{preceding\_sibling} :: */ \\
 & \quad (\texttt{descendant} :: \texttt{NameTest}\ \texttt{union}\ \texttt{self} :: \texttt{NameTest}) \\
\texttt{if T then R else S} & \equiv .[\texttt{T}]/\texttt{R}\ \texttt{union}\ .[\texttt{not T}]/\texttt{S}
\end{array}$$

Definable node expressions:

$$\begin{array}{ll}
\texttt{true}() & \equiv\ . \\
\texttt{false}() & \equiv\ () \\
\texttt{exists}(\texttt{R}) & \equiv\ \texttt{R} \\
\texttt{empty}(\texttt{R}) & \equiv\ \texttt{not}\ (\texttt{R}) \\
\texttt{some \$i in R satisfies T} & \equiv\ \texttt{for \$i in R return} .[\texttt{T}] \\
\texttt{every \$i in R satisfies T} & \equiv\ \texttt{not}\ (\texttt{for \$i in R return} .[\texttt{not}\ (\texttt{T})])
\end{array}$$

Fig. 3 Definable XPath 2.0 operations

$$\begin{array}{ll}
(\texttt{Axis} :: \texttt{NameTest})^{\smile} & \equiv \texttt{self} :: \texttt{NameTest/Axis}^{\smile} :: * \\
.^{\smile} & \equiv . \\
\texttt{\$i}^{\smile} & \equiv .\,[.\,\texttt{is}\,\texttt{\$i}]\,/\top \\
()^{\smile} & \equiv () \\
(R/T)^{\smile} & \equiv T^{\smile}/R^{\smile} \\
(R\ \texttt{union}\ T)^{\smile} & \equiv R^{\smile}\ \texttt{union}\ T^{\smile} \\
(R\ \texttt{intersect}\ T)^{\smile} & \equiv R^{\smile}\ \texttt{intersect}\ T^{\smile} \\
(R\ \texttt{except}\ T)^{\smile} & \equiv R^{\smile}\ \texttt{except}\ T^{\smile} \\
(R[X])^{\smile} & \equiv .\,[X]/R^{\smile}
\end{array}$$

where $\texttt{Axis}^{\smile}$ is the converse of `Axis` (i.e., $\texttt{child}^{\smile} = \texttt{parent}$, etc.). The reader may verify that $R^{\smile}$ indeed defines the converse of R, i.e., $(x, y) \in [\![R^{\smile}]\!]^g$ iff $(y, x) \in [\![R]\!]^g$.

2.4 Undecidable Extensions

Because we want to axiomatize query equivalence on finite models it is necessary that query equivalence is decidable. Here we show how value comparisons and functions as `position()`, `last()`, and `count()` can lead to undecidability.

Positional Information XPath supports reference to positional information in predicates, via the functions `position()`, `last()`, and `count()`. For example, `child::*[position()=1]` and `child::*[position()=last()]` (commonly abbreviated as `child::*[1]` and `child::*[last()]`, respectively) return the first and last child of the context node, in document order. For the formal semantics of these predicates, see [11].

The simplest type of positional predicates, of the form R`[position()=`k`]` or R`[position()=last()-`k`]`, ("return the k-th, respectively, k-but-last, node in document order that can be reached from the context node via R") is quite harmless. It can be eliminated at the cost of an exponential blowup, using the following equivalences:

$$\begin{array}{ll}\texttt{R[position()=1]} & \equiv \texttt{R except (R/}\ll\texttt{)} \\ \texttt{R[position()=}k\texttt{+1]} & \equiv \texttt{(R intersect (R[}k\texttt{]/}\ll\texttt{))[position()=1]}\end{array}$$

and symmetrically for R`[last()-`k`]`. These equivalences could in principle be added to the axiomatization we will give below in order to obtain completeness w.r.t. this enrichment of the language.[1]

More advanced use of positional or counting information quickly makes the logic undecidable:

Theorem 2 *Core XPath extended with expressions of any of the following forms is undecidable for query equivalence*:

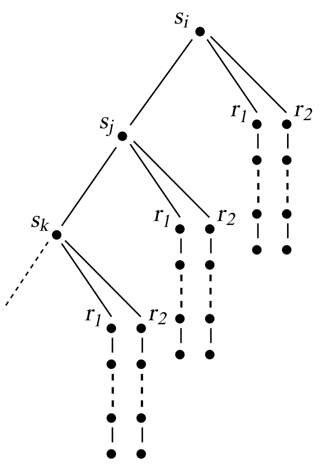
- R`[count(`R'`)` $=$ `count(`R''`)]`, *or*
- R`[position()` $=$ `last()/2]`, *or*
- R`[position()` $=$ `count(`R'`)]`.

Proof We sketch a reduction to the undecidable halting problem for two-register machines (2RM) [7]. We follow the definition given in [3]. 2RMs are similar to Turing machines but instead of a tape they have two registers r_1, r_2. Each register contains a natural number. A 2RM is programmed by a numbered sequence $I_0, I_1, \ldots, I_l$ of instructions. It indicates that M is to execute instruction I_i with registers r_1 and r_2 containing m and n, respectively. Each instruction I_i is either an addition or a subtraction. An addition has the form (rg, j), with rg a register number and $j \leq l$ an instruction number. Its semantics is: *add one to register rg and move to instuction I_j*. A subtraction has the form (rg, j, k) with rg a register number and $j, k \leq l$ instruction numbers. Its semantics is: *if content of register rg is zero then move to instruction I_j, otherwise subtract one from register rg and move to instruction I_k*. Each 2RM has also a designated starting instruction, which we may assume to be I_0, and final instruction I_f $(f \leq l)$. An instantaneous description (ID) of a 2RM M is a triple (i, m, n), where $i \leq l$ is an instruction number and $m, n \geq 0$ are natural numbers representing the content of registers r_1 and r_2. The *halting problem for 2RMs* is to determine for a given 2RM whether there is a run of the machine starting in the ID $(0, 0, 0)$ and ending in the ID $(f, 0, 0)$. This problem is known to be undecidable.

Following [3], we represent runs of 2RMs as XML documents. Our representation, depicted in Fig. 4, is based on the representation used in the proof of [3, Theorem 7.9]. Each node on the left most branch (i.e., satisfying $\neg(\uparrow^+ / \leftarrow)$) represents an ID (i, j, k) of the machine. The node tag s_i encodes the instruction number i, while the register contents j and k are represented by the number of descendants of the r_1 and r_2 child, respectively. For each 2RM M, we can create a node expression ϕ_M that is satisfied precisely in the roots of those XML documents that encode runs of M

[1]Note, however, that certain axiom schemes that are valid in the language without positional predicates, such as $(R/S)[X] \equiv R/(S[X])$, would no longer be valid in the extended language.

Fig. 4 XML encoding of a two-register machine run



starting in $(0,0,0)$ and ending in $(f,0,0)$. Hence, M halts iff ϕ_M is not equivalent to `()`.

The predicates in the theorem are used to express that, in going from one ID to the next, the content of the registers is appropriately related. For example, the following node expressions, evaluated in a node on the left most branch, test that the content of register r_1 is the same in the current ID and in the next one:

- $\texttt{count(}\downarrow[r_1]/\downarrow^+\texttt{)} = \texttt{count(}\downarrow_{\mathsf{next}}/\downarrow[r_1]/\downarrow^+\texttt{)}$, or
- $\texttt{((}\downarrow_{\mathsf{next}} \cup .)/\downarrow[r_1]/\downarrow^+\texttt{)[position()} = \texttt{last()/2])[}\neg\downarrow\texttt{]}$, or
- $\texttt{((}\downarrow[r_1]/\downarrow^+\texttt{) [position()} = \texttt{count(}\uparrow/\downarrow_{\mathsf{next}}/\downarrow[r_1]/\downarrow^+\texttt{)])[}\neg\downarrow\texttt{]}$

where $\downarrow_{\mathsf{next}}$ is shorthand for $\downarrow[\neg\leftarrow]$). Similarly, we can test that the value of a register is incremented or decremented using such predicates. □

Comparison Operators XPath 2.0 has three sets of operators to compare nodes. Here we just discuss the three types of *equality*: $=$, `eq` and `is`. Each takes two path expressions as input. The operators $=$ and `eq` compare the data values of input nodes (in XPath terminology, the *atomization* of the input), while `is` compares node identity. a `eq` b is true if the nodes a and b have the same data value, and a `is` b is true if a and b are the same node. $=$ can be defined in terms of `eq`: $R = S$ iff `some $r in` R `satisfies some $s in` S `satisfies $r eq $s` (recall from Fig. 3 that `some` can be expressed in terms of `for`). Node equality is term definable in XPath 2.0 using intersection (see axiom Eq1 in Fig. 9). Data value equality quickly leads to undecidability: already in the context of Core XPath [10]; or in first order logic with three variables having only the `child` relation interpreted on unary trees [6]. Both proofs use Post's correspondence problem.

QT1.	$\forall xyz(x < y \wedge y < z \rightarrow x < z)$	$<$ is transitive
QT2.	$\neg\exists x(x < x)$	$<$ is irreflexive
QT3.	$\forall xy(x < y \rightarrow \exists z(x <_{imm} z \wedge z \leq y))$	immediate children
QT4.	$\exists x\forall y\neg(y < x)$	there is a root
QT5.	$\forall xyz(x < z \wedge y < z \rightarrow x \leq y \vee y \leq x)$	linearly ordered ancestors
QT6.	$\forall xyz(x \prec y \wedge y \prec z \rightarrow x \prec z)$	$\prec$ is transitive
QT7.	$\neg\exists x(x \prec x)$	$\prec$ is irreflexive
QT8.	$\forall xy(x \prec y \rightarrow \exists z(x \prec_{imm} z \wedge z \preceq y))$	immediately next sibling
QT9.	$\forall x\exists y(y \preceq x \wedge \neg\exists z(z \prec y))$	there is a least sibling
QT10.	$\forall xy((x \prec y \vee y \prec x) \leftrightarrow$ $(\exists z(z <_{imm} x \wedge z <_{imm} y) \wedge x \neq y))$	$\prec$ linearly orders *siblings*
QT11.	$\forall xy(x = y \vee x < y \vee y < x \vee$ $\exists x'y'(x' \leq x \wedge y' \leq y \wedge (x' \prec y' \vee y' \prec x')))$	connectedness
QT-Ind.	$\forall x(\forall y(x{<<}y \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow \forall x.\phi(x)$	induction scheme

where

$x <_{imm} y$ is shortland for $x < y \neg\exists z(x < z \wedge z < y)$,
$x \prec_{imm} y$ is shortland for $x \prec y \neg\exists z.(x \prec z \wedge z \prec y)$, and
$x \ll y$ is shortland for $x < y \vee \exists x'y'(x' \leq x \wedge y' \leq y \wedge x' \prec y')$

Fig. 5 Axioms of the FO theory of finite trees

3 Relations with other Languages

We discuss the relation of Core XPath 2.0 with first order logic and with Tarski's relation algebras. The connections between these formalisms will be exploited in our proofs later on.

3.1 First Order Logic

XPath 2.0 was designed to be expressively complete for first-order queries [16]. More precisely, we mean here first-order logic in the signature with binary relation symbols $<$ and $\prec$ denoting the *descendant* and *following sibling* relations, and arbitrarily many unary predicates for the node tags. The expression $x < y$ means that y is a descendant of x, and $x \prec y$ means that y is a following sibling of x. It is easy to see that there are linear translations between this first-order language and Core XPath 2.0. The `for`-loop and the variables are not even needed for expressive completeness: even just Core XPath 1.0 extended with the `except` operator is already expressively complete for this first-order language [19]. Whether this holds in the presence of data value comparisons depends on a longstanding open problem in finite model theory, namely whether FO has a finite variable property on finite ordered structures [8].

An axiomatization of the first-order theory of finite node-labeled sibling ordered trees (in the signature described above) was given in [2], together with a rather involved completeness proof. In Appendix we provide a short alternative completeness proof for the axiomatization given in Fig. 5. First of all, note that these axioms define, among all finite structures, those that are trees:

Theorem 3 *A finite structure $M = (\mathrm{dom}, <, \prec, P_1, \ldots, P_n)$ satisfies all axioms in Fig. 5 iff it is isomorphic to a finite node-labeled sibling ordered tree.*

The proof is left as an exercise to the reader.

Note that a structure satisfying the axioms in Fig. 5 is not necessarily a finite tree, as it may be infinite. Still, the axioms in Fig. 5 completely axiomatize the theory of finite trees:

Theorem 4 *The FO theory of finite node-labeled sibling ordered trees is completely axiomatized by the axioms and axiom scheme in Fig. 5.*

The proof is given in Appendix.

Next, we give a more precise statement of the first-order expressive completeness of Core XPath 2.0, as we will need to refer to it later on.

Theorem 5

1. *For every FO formula $\phi(x, y)$ in two free variables there is a variable-free Core XPath* 2.0 PathExpr *R such that the two define the same binary relation in each tree.*
2. *For every FO formula $\phi(x)$ in one free variable there is a variable-free Core XPath* 2.0 TestExpr *X such that the two define the same set of nodes in each tree.*

In fact, the above hold not only for trees, but even on the more general class of structures defined by the FO axioms QT1–QT11 in Fig. 5.

Proof The first item was proved in [19, Corollary 3.5], whose proof spans Sect. 4 of [19]. Although the result was stated for finite trees only, a careful reading of Sect. 4 of [19] shows that it holds on the more general class of structures defined by the axioms QT1–QT11. Indeed, finiteness or well-foundedness is not used in the proof in any crucial way. The arguments *do* use the fact that every two nodes have a least common ancestor but this follows from the given axioms (in particular, the axiom QT11).

The second item follows directly from the first (cf. also the proof of Corollary 3.3 in [19]): given $\phi(x)$, let R be the variable-free Core XPath 2.0 PathExpr that defines the same binary relation as $\phi(x) \wedge y = y$. Then R—taken as a TestExpr—defines the same set of nodes as $\phi(x)$. □

3.2 Relation Algebra

The variable free fragment of Core XPath 2.0 (which, as we mentioned earlier, already has the full expressive power of Core XPath 2.0) is closely related to Tarski's algebra of binary relations ("Relation Algebra") [20] cf., also [12, 14]. This elegant and purely algebraic language consists of

- a set *RVAR* of atomic expressions denoting binary relations (over some set)
- constants $\top$, $\bot$ and ., denoting the total relation, the empty relation and the identity relation, respectively

Fig. 6 Valid equivalences of relation algebra

$RA1.$	$(R/S)/T$	$\equiv$	$R/(S/T)$
$RA2.$	$R/.$	$\equiv$	R
$RA3.$	$(R\cup S)/T$	$\equiv$	$(R/T)\cup(S/T)$
$RA4.$	$(R\cup S)^{\smile}$	$\equiv$	$R^{\smile}\cup S^{\smile}$
$RA5.$	$(R/S)^{\smile}$	$\equiv$	$S^{\smile}/R^{\smile}$
$RA6.$	$(R^{\smile})^{\smile}$	$\equiv$	R
$RA7.$	$S/(-(S^{\smile}/R))$	$\subseteq$	$-R$

- operators for taking *union* ($\cup$), *intersection* ($\cap$), *complement* ($-$), *composition* ($/$) and *converse* ($\cdot^{\smile}$).

In addition, we use diff as a shorthand for $-(.)$, i.e., the inequality relation.

As the reader can observe, the main syntactic differences between the variable free fragment of Core XPath and Relation Algebra are (i) Relation Algebra has a converse operator, while in Core XPath 2.0 each individual expression has a definable converse, (ii) Relation Algebra uses a unary complementation operator, whereas Core XPath 2.0 uses relative complementation, (iii) Relation Algebra includes a constant $\top$ denoting the universal relation, whereas in Core XPath 2.0 the universal relation is defined by `(ancestor::* union self::*)/(descendant::* union self::*)`, (iv) Core XPath 2.0 features *predicates*, which are not present in Relation Algebras.

These differences in syntax are mostly cosmetic. The most important difference between Relation Algebra and Core XPath 2.0 lies in the *semantics*: Core XPath 2.0 is interpreted on finite trees whereas Relation Algebra is traditionally concerned with arbitrary structures consisting of a domain and an interpretation for each atomic binary relation symbol. Still, in this paper, we will make important use of known results about Relation Algebra.

A list of valid equivalences of relation algebra is given in Fig. 6. Each of these equivalences expresses that on every structure, all pairs of elements in the domain of the structure standing in the first relation also stand in the second relation and vice versa. We use $\alpha\subseteq\beta$ as a shorthand for the equivalence $\beta\equiv\beta\cup\alpha$. Tarski [20] originally proposed these equivalences as axioms, and conjectured that, combined with any complete set of axioms of Boolean algebra (for $\cap,\cup,-,\top$), they would constitute a complete axiomatization of the equational theory of relation algebra. However, this turned out not to be the case, a result of Lyndon [17, 18]. A finite non-representable algebras satisfying the axioms was found by McKenzie, see [21]. A large body of subsequent work studied axiomatizations and axiomatizability for relation algebra, an overview of which can be found in [15]. For us, the following positive result will be of particular relevance (cf. [24, Sect. 3.5.3], and also [26, 28] for related results).

Theorem 6 (Venema) *Fix a countably infinite set of relational variables RVAR. Two relation algebra expressions α,β over RVAR are equivalent iff their equivalence is provable from the axioms in Fig. 6 and the axioms of Boolean algebra using the standard rules of equational logic and the following non-standard inference rule* (Ven):

$$\textit{If } (R\cap -((\mathsf{diff}/R/\top)\cup(\top/R/\mathsf{diff})))\subseteq\alpha,\quad \textit{for } R\in RVAR \textit{ not occurring in } \alpha,$$

then $\alpha \equiv \top$.

Here, by the *standard rules of equational logic*, we mean the reflexivity rule (for every expression α, $\alpha \equiv \alpha$), the symmetry rule (if $\alpha \equiv \beta$ then $\beta \equiv \alpha$), the transitivity rule (if $\alpha \equiv \beta$ and $\beta \equiv \gamma$ then $\alpha \equiv \gamma$) and the rule of substituting equals by equals (if $\alpha \equiv \beta$ and δ is obtained from γ by replacing some occurrences of α by β, then $\gamma \equiv \delta$).

Venema's proof is based on a Henkin-style model construction. Recall that, in a standard Henkin-style completeness proof for first-order logic, a structure is constructed out of equivalence classes of constants. In the case of relation algebra, the structure is constructed out of special relation algebra expressions which we will call *nominals*. Intuitively, a nominal is a relation algebra expression denoting a binary relation consisting of at most one pair, and (Ven) allows us to assume the existence of such nominals. Formally, let Σ be any set of equivalances between relation algebra expressions. We say that an equivalence $\alpha \equiv \beta$ *follows from* Σ, also denoted by $\Sigma \vdash \alpha \equiv \beta$, if it can be obtained from Σ plus the axioms using the standard rules of equational logic. We say that a relation algebra expression α is a Σ-*nominal* if

$$\Sigma \vdash \alpha \subseteq -((\mathsf{diff}/\alpha/\top) \cup (\top/\alpha/\mathsf{diff})).$$

The intuition is that the binary relation denoted by α consists of at most one pair. In particular, note that the expression $R \cap -((\mathsf{diff}/R/\top) \cup (\top/R/\mathsf{diff}))$ from the Ven-rule is a nominal. Thus we can understand the Ven-rule (in contraposition) as saying that if $\alpha \not\equiv \top$, then either $\alpha \equiv \bot$ or there exists a nominal β such that $\beta \subseteq -\alpha$.

We say that Σ *provides Henkin witnesses* (with respect to a given set of relational variables *RVAR*) if for every *RVAR*-relation algebra expression α there is a *RVAR*-relation algebra expression β such that

1. β is a Σ-nominal
2. $\Sigma \vdash \beta \subseteq \alpha$
3. $\Sigma \vdash \top/\alpha/\top \equiv \top/\beta/\top$.

Intuitively, the third item says that if α defines a non-empty relation, β does too.

Venema's proof of Theorem 6 centers around the following two lemmas.

Lemma 7 *Fix a countably infinite set of relational variables RVAR. For any two relation algebra expressions α, β over RVAR, if $\alpha \equiv \beta$ is not provable from the axioms in Fig. 6 and the axioms of Boolean algebra using the rules of equational logic and* (Ven), *then there is a set of equivalences Σ over RVAR providing Henkin witnesses w.r.t. RVAR and such that $\Sigma \not\vdash \alpha \equiv \beta$.*

Lemma 8 *Let RVAR be any set of relational variables and Σ a set of equivalences over RVAR. If Σ provides Henkin witnesses w.r.t. RVAR, then for any relation algebra expressions α, β over RVAR, $\Sigma \vdash \alpha \equiv \beta$ iff α and β are equivalent on structures satisfying all equivalences in Σ.*

Theorem 6 directly follows from these two lemmas. The second lemma will play a crucial role in our completeness proofs for Core XPath 2.0.

4 A Complete Equational Calculus

In this section, we will give a complete equational axiomatization of equivalence in Core XPath 2.0. First, we axiomatize the variable free fragment, and then we show how the axiomatization can be extended to the full language of Core XPath 2.0. We will make use of the shorthand notations introduced in Sect. 2.3.

We do not enforce that, as in real XML document trees, each node is labeled by exactly one node label, as this can always be enforced: if R, S are path expressions containing node labels $p_1, \dots, p_n$, and R', S' are obtained from R, S by uniformly replacing each node label p_i by $p_i \wedge \bigwedge_{j \neq i} \neg p_j$, then R' and S' are equivalent on XML documents with multiple node labels iff R and S are equivalent on XML documents in which each node has a single label.

We reiterate the chosen signature from Sect. 2.3. The atomic symbols are all Step productions from Fig. 1. Those of the form `axis` :: $*$ are abbreviated by arrows. In the axioms, the inductively defined converse function $(\cdot)^{\smile}$ is used. At the base level, it relates the arrows $((\downarrow^+)^{\smile} = \uparrow^+$ and $(\uparrow^+)^{\smile} = \downarrow^+$, etc.).

4.1 The Variable Free Fragment

Recall that the *variable free fragment* of Core XPath 2.0 is the fragment without variables, `for` and node comparison tests. As we saw in Sect. 3.2, it is closely related to Tarski's Relation Algebra [13, 20].[2] The calculus we will present essentially extends a known complete axiomatization of Relation Algebra with axioms that capture the special properties of finite node-labeled sibling-ordered trees ("finite trees", for short).

Our axioms are given in Fig. 7.[3] In this table, and in what follows, R, S, T range over (variable free) path expressions, and X, Y range over (variable free) test expressions (thus, most of the "axioms" are actually schemes that have infintely many instantiations). We use $R \subseteq S$ as shorthand for $S \equiv S \cup R$. A few words about the axioms. They are divided into four groups. The Boolean axioms BA1–BA11 are standard. From Tarski's axioms RA1–RA7 we could remove RA4–RA6 because they are about the converse operator and that is a defined operator with us, cf. Lemma 10. The axioms for the predicates show that predicates are just syntactic sugar. Axioms Tr1–Tr11 and *Ind* are direct translations of the first-order axioms in Fig. 5. In particular, the *Ind* axiom captures the fact that, whenever there is a R-path from a node d to a node e, then there is a *last* node e' in document order for which it is the cases

[2]We will use algebraic results from [13] in our proofs, so we relate the notation used there with our XPath notation. According to [13] Def. 5.3.1, a relation algebra (RA) is an algebra $(A, +, \cdot, -, 0, 1, ;, {}^{\smile}, 1')$, with $(A, +, \cdot, -, 0, 1)$ a Boolean Algebra, ; a binary operation on A, ${}^{\smile}$ a unary operation on A and $1' \in A$. These operators are related to our XPath operators as follows: $+$ and $\cdot$ denote `union` and `intersect`, respectively. $-$ is complementation with respect to A. Thus x `except` $y \equiv x \cdot -y$, and $-y \equiv \top$ `except` y. 0 and 1 denote the Boolean constants false and true, respectively, in our notation $\bot$ and $\top$. ";" denotes the composition operator for which XPath uses the symbol "/". The operator $(\cdot)^{\smile}$ denotes inverse in both systems. The identity contant $1'$ is denoted by the period "." in XPath.

[3]The induction axiom *Ind* in Fig. 7 is different from axiom we used in the earlier version of this paper that was published in the ICDT 2007 conference proceedings. The old version contained a mistake, and the corrected version of the axiom we use here is also more easy to read.

Axioms of Boolean Algebra and Relation Algebra

BA1.	$R\ \texttt{union}\ (S\ \texttt{union}\ T)$	$\equiv$	$(R\ \texttt{union}\ S)\ \texttt{union}\ T$
BA2.	$R\ \texttt{intersect}\ (S\ \texttt{intersect}\ T)$	$\equiv$	$(R\ \texttt{intersect}\ S)\ \texttt{intersect}\ T$
BA3.	$R\ \texttt{union}\ S$	$\equiv$	$S\ \texttt{union}\ R$
BA4.	$R\ \texttt{intersect}\ S$	$\equiv$	$S\ \texttt{intersect}\ R$
BA5.	$R\ \texttt{union}\ (S\ \texttt{intersect}\ T)$	$\equiv$	$(R\ \texttt{union}\ S)\ \texttt{intersect}\ (R\ \texttt{union}\ T)$
BA6.	$R\ \texttt{intersect}\ (S\ \texttt{union}\ T)$	$\equiv$	$(R\ \texttt{intersect}\ S)\ \texttt{union}\ (R\ \texttt{intersect}\ T)$
BA7.	$R\ \texttt{union}\ (R\ \texttt{intersect}\ S)$	$\equiv$	R
BA8.	$R\ \texttt{intersect}\ (R\ \texttt{union}\ S)$	$\equiv$	R
BA9.	$R\ \texttt{union}\ (\top\ \texttt{except}\ R)$	$\equiv$	$\top$
BA10.	$R\ \texttt{intersect}\ (\top\ \texttt{except}\ R)$	$\equiv$	$\bot$
BA11.	$R\ \texttt{intersect}\ (\top\ \texttt{except}\ S)$	$\equiv$	$R\ \texttt{except}\ S$
RA1.	$(R/S)/T$	$\equiv$	$R/(S/T)$
RA2.	$R/.$	$\equiv$	R
RA3.	$(R\ \texttt{union}\ S)/T$	$\equiv$	$(R/T)\ \texttt{union}\ (S/T)$
RA7.	$(S/(\top\ \texttt{except}\ (S^{\smile}/R)))$	$\subseteq$	$\top\ \texttt{except}\ R$

Axioms for eliminating predicates

Pred1.	$R[X\ \texttt{and}\ Y]$	$\equiv$	$R[X][Y]$
Pred2.	$R[X\ \texttt{or}\ Y]$	$\equiv$	$R[X]\ \texttt{union}\ R[Y]$
Pred3.	$R[\texttt{not}(X)]$	$\equiv$	$R\ \texttt{except}\ R[X]$
Pred4.	$R[S]$	$\equiv$	$R/((S/\top)\ \texttt{intersect}\ .)$

Axioms for finite trees:

Tr1.	$\downarrow^{+} / \downarrow^{+}$	$\subseteq$	$\downarrow^{+}$
Tr2.	$\downarrow^{+}\ \texttt{intersect}\ \uparrow^{+}$	$\equiv$	$\bot$
Tr3a.	$\downarrow^{+}$	$\equiv$	$\downarrow\ \texttt{union}\ (\downarrow / \downarrow^{+})$
Tr3b.	$\downarrow$	$\equiv$	$\downarrow^{+}\ \texttt{except}\ (\downarrow^{+} / \downarrow^{+})$
Tr4.	$.[\uparrow]$	$\equiv$	$.[\uparrow^{+}[\texttt{not}(\uparrow)]]$
Tr5.	$\downarrow^{+} / \uparrow^{+}$	$\equiv$	$\downarrow^{+}[\downarrow]\ \texttt{union}\ .[\downarrow]\ \texttt{union}\ .[\downarrow]/\uparrow^{+}$
Tr6.	$\rightarrow^{+} / \rightarrow^{+}$	$\subseteq$	$\rightarrow^{+}$
Tr7.	$\rightarrow^{+}\ \texttt{intersect}\ \leftarrow^{+}$	$\equiv$	$\bot$
Tr8a.	$\rightarrow^{+}$	$\equiv$	$\rightarrow\ \texttt{union}\ (\rightarrow / \rightarrow^{+})$
Tr8b.	$\rightarrow$	$\equiv$	$\rightarrow^{+}\ \texttt{except}\ (\rightarrow^{+} / \rightarrow^{+})$
Tr9.	$.[\leftarrow]$	$\equiv$	$.[\leftarrow^{+}[\texttt{not}(\leftarrow)]]$
Tr10.	$\rightarrow^{+}\ \texttt{union}\ \leftarrow^{+}$	$\equiv$	$(\uparrow / \downarrow)\ \texttt{except}\ .$
Tr11.	$.\ \texttt{union}\ \uparrow^{+}\ \texttt{union}\ \downarrow^{+}\ \texttt{union}\ (\uparrow^{*}/\rightarrow^{+}/\downarrow^{*})\ \texttt{union}\ (\uparrow^{*}/\leftarrow^{+}/\downarrow^{*})$	$\equiv$	$\top$
Ind.	$.[R]$	$\equiv$	$.[R\ \texttt{except}\ R/\texttt{>>}]$

Axioms for tag-names:

Tag1.	`Axis::NameTest`	$\equiv$	`Axis::*/self::NameTest`
Tag2.	`self::NameTest`	$\subseteq$	.

Fig. 7 Axioms for Core XPath 2.0 query equivalence

that there is an R-path from d to e'. It holds by virtue of the finiteness of the XML document trees, and is analogous to the induction axiom in arithmetic, which states (in one of its equivalent formulations) that every non-empty set of natural numbers has a smallest element.[4] The axioms in the last group handle the particular type of atomic symbols in XPath.

Keep in mind that these axioms describe *relative path equivalence* (as opposed to *equivalence when evaluated at the root*). We leave it to the reader to check that axioms in Fig. 7 are indeed sound.

We say that R is *provably equivalent* to S in our calculus (notation: $R \equiv^* S$), if the equivalence of R and S can be proved from substitution instances of the axioms using the *transitivity, symmetry and reflexivity of* $\equiv$ and *replacement of equals by equals* (if $R \equiv S$ and T' is obtained from T by replacing some occurrences of R by S, then $T \equiv T'$). Likewise, we use $\subseteq^*$ for provable *containment*. We will usually abstract away from Boolean reasoning: since our axiomatization includes a complete set of axioms for Boolean Algebra, we know that all valid Boolean equivalences are derivable, and hence we will use them at liberty in the proofs.

Theorem 9 (Completeness) *For any two equivalent variable-free expressions R and S, $R \equiv^* S$.*

In the remainder of this section, we prove Theorem 9. We first establish two useful Boolean facts and three preliminary lemmas showing that certain equalities are derivable. Then we can smoothly prove completeness.

It is sometimes easier to prove mutual containment of two expressions than equivalence. Fortunately, the two are equivalent:

Fact 1 *For all expressions R and S, $R \equiv^* S$ iff both $R \subseteq^* S$ and $S \subseteq^* R$.*

Proof This follows from the fact that we have all axioms of Boolean Algebra. However, to be complete we will also give a direct proof here. If $R \equiv^* S$, then, by axiom BA7, $R \equiv^* R\ \texttt{union}\ (S\ \texttt{intersect}\ (S\ \texttt{union}\ S))$, and hence, by axiom BA8, $R \equiv^* R\ \texttt{union}\ S$. Similarly, we can show that $S \equiv^* S\ \texttt{union}\ R$. Conversely, if both inclusions are derivable, then $S \equiv^* S\ \texttt{union}\ R$ and $R \equiv^* R\ \texttt{union}\ S$, hence, by transitivity and axiom BA4, $R \equiv^* S$. □

In proofs of containment, the following fact can be very useful. Call a subexpression of an expression R positive (negative), if it occurs under the scope of an even (odd) number of polarity switching operators. Here, by polarity switching operators we mean `not` and `except`, and the latter is only counted as polarity switching with respect to the *second* argument.

[4] The relationship between *Ind* and the induction scheme *QT-Ind* from Fig. 5 may be easier to see if one takes the contraposition of the latter, after substituting $\neg\phi(x)$ for $\psi(x)$, i.e., $\exists x.\psi(x) \rightarrow \exists x.(\psi(x) \wedge \neg\exists y.(x \ll y \wedge \psi(y)))$.

RA4.	$(R\ \texttt{union}\ S)^{\smile}$	$\equiv$	$R^{\smile}\ \texttt{union}\ S^{\smile}$
RA5.	$(R/S)^{\smile}$	$\equiv$	$S^{\smile}/R^{\smile}$
RA6.	$(R^{\smile})^{\smile}$	$\equiv$	R
RA8.	$R/(S\ \texttt{union}\ T)$	$\equiv$	$R/S\ \texttt{union}\ R/T$
RA9.	$./R$	$\equiv$	R
RA10.	$R/\bot$	$\equiv$	$\bot$
RA11.	$\top/\top$	$\equiv$	$\top$
RA12.	$.^{\smile}$	$\equiv$	$.$
RA13.	$\bot^{\smile}$	$\equiv$	$\bot$
RA14.	$\top^{\smile}$	$\equiv$	$\top$
RA15.	$(R\ \texttt{intersect}\ S)^{\smile}$	$\equiv$	$R^{\smile}\ \texttt{intersect}\ S^{\smile}$
RA16.	$(R\ \texttt{except}\ S)^{\smile}$	$\equiv$	$R^{\smile}\ \texttt{except}\ S^{\smile}$
RA17.	$(\top\ \texttt{except}\ R/S)/S^{\smile}$	$\subseteq$	$(\top\ \texttt{except}\ R)$
RA18.	$(R\ \texttt{intersect}\ .)/(R\ \texttt{intersect}\ .)$	$\equiv$	$(R\ \texttt{intersect}\ .)$
RA19.	$(R\ \texttt{intersect}\ .)^{\smile}$	$\equiv$	$(R\ \texttt{intersect}\ .)$
RA20.	$((R/\top)\ \texttt{intersect}\ .)/\top$	$\equiv$	$R/\top$
Pred5.	$R/.[S]$	$\equiv$	$R[S]$
Pred6.	R/S	$\equiv$	$R[S]/S$
Pred7.	$R[S[T]]$	$\equiv$	$R[S/T]$
Pred8.	$(R[S])^{\smile}$	$\equiv$	$.[S]/R^{\smile}$
Pred9.	$\top[S]/R$	$\equiv$	$\top[R^{\smile}/S]$
Pred10.	$R[S\ \texttt{and}\ \texttt{not}(T)]$	$\subseteq$	$R[(S\ \texttt{except}\ T)]$
Pred11.	$\top[\top[R]]$	$\equiv$	$\top/R/\top$

Fig. 8 Equivalences derivable from the axioms in Fig. 7

Fact 2 (Monotonicity ([13] Lemmas 5.3.3.(iv) and 5.3.5.(i))) *Suppose $R \subseteq^* S$, and let T' be obtained from T by replacing some positive occurrences of R by S, and/or replacing some negative occurrences of S by R. Then $T \subseteq^* T'$.*

In the completeness proof we will use the derivable equivalences listed in Fig. 8.

Lemma 10 *All equivalences and containments in Fig. 8 are derivable.*

Proof *RA4* and *RA5* follow immediately from the definition of the function $(\cdot)^{\smile}$. The same holds for *RA6* if we consider expressions that do not contain predicates. For expressions R containing predicates, it is not immediately obvious that *RA6* is derivable, since $((R[X])^{\smile})^{\smile}$ is not by definition identical to $R[X]$. However, by *Pred1–4*, R is provably equivalent to an expression R' that contains no predicates, and therefore $(R^{\smile})^{\smile} \equiv^* ((R')^{\smile})^{\smile} \equiv^* R' \equiv^* R$. For all RA$n$ with $n \geq 8$, the proof is in [13], with these correspondences:

RA8	5.3.3.(i)	RA13	5.3.3.(vi)	RA17	5.3.3.(ix)
RA9	5.3.3.(iii)	RA14	5.3.3.(vii)	RA18	5.3.5.(x)
RA10	5.3.5.(v)	RA15	5.3.3.(viii)	RA19	5.3.5.(ix)
RA11	5.3.3.(iii) and Fact 2	RA16	5.3.3.(viii)	RA20	5.3.5.(xiii)
RA12	5.3.3.(ii).				

The remaining equivalences are proved below.

► *Pred5.*

$$\begin{array}{ll} & R/.[S] \\ \equiv^*_{(Pred4)} & R/./((S/\top) \cap .) \\ \equiv^*_{(RA2)} & R/((S/\top) \cap .) \\ \equiv^*_{(Pred4)} & R[S] \end{array}$$

► *Pred6.* The proof proceeds in two steps. First,

$$\begin{array}{ll} & R/S \\ \equiv^*_{(RA2)} & R/./S \\ \equiv^*_{(Boolean)} & R/\big(((S/\top)\ \texttt{intersect}\ .) \\ & \texttt{union}\ \ ((\top\ \texttt{except}\ \ S/\top)\ \texttt{intersect}\ \ .)\big)/S \\ \equiv^*_{(RA3)} & R/((S/\top)\ \texttt{intersect}\ \ .)/S\ \texttt{union} \\ & R/((\top\ \texttt{except}\ \ S/\top)\ \texttt{intersect}\ \ .)/S \end{array}$$

Next, we will show that the second half of the last expression is provably equivalent to $\bot$, from which the result follows. By Boolean reasoning and monotonicity, $(R \cap S)/T \subseteq^* (R/T) \cap (S/T)$. Hence,

$$\begin{array}{ll} & R/((\top\ \texttt{except}\ \ S/\top)\ \texttt{intersect}\ \ .)/S \\ \subseteq^* & R/\big((\top\ \texttt{except}\ \ S/\top)/S\ \texttt{intersect}\ \ (./S)\big) \\ \subseteq^*_{(Monotonicity,\ RA9)} & R/\big((\top\ \texttt{except}\ \ S/S^{\smile})/S\ \texttt{intersect}\ \ S\big) \\ \subseteq^*_{(RA17)} & R/\big((\top\ \texttt{except}\ \ S)\ \texttt{intersect}\ \ S\big) \\ \subseteq^*_{(Boolean)} & R/\bot \\ \subseteq^*_{(RA10)} & \bot \end{array}$$

The reverse inclusion, i.e., $R[S]/S \subseteq^* R/S$, holds again by Boolean reasoning and monotonicity.

► *Pred7.*

$$\begin{array}{ll} & R[S[T]] \\ \equiv^*_{(Pred4)} & R/((S/((T/\top)\ \texttt{intersect}\ \ .)/\top)\ \texttt{intersect}\ \ .) \\ \equiv^*_{(RA20)} & R/((S/T/\top)\ \texttt{intersect}\ \ .) \\ \equiv^*_{(Pred4)} & R[S/T] \end{array}$$

► *Pred8.*

$$\begin{array}{ll} & R[S]^{\smile} \\ \equiv^*_{(Pred4)} & (R/((S/\top)\ \texttt{intersect}\ \ .))^{\smile} \\ \equiv^*_{(RA5)} & ((S/\top)\ \texttt{intersect}\ \ .)^{\smile}/R^{\smile} \\ \equiv^*_{(RA19)} & ((S/\top)\ \texttt{intersect}\ \ .)/R^{\smile} \\ \equiv^*_{(RA9)} & ./((S/\top)\ \texttt{intersect}\ \ .)/R^{\smile} \\ \equiv^*_{(Pred4)} & .[S]/R^{\smile} \end{array}$$

▶ *Pred9*. We first show that $\top/R \equiv^* \top[R^{\smile}]$: In one direction,

$$\begin{array}{ll}
 & \top/R \\
\equiv^*_{(RA2)} & \top/R/. \\
\equiv^*_{(RA6)} & \top/(R/.)^{\smile\smile} \\
\equiv^*_{(RA5)} & \top/(.^{\smile}/R^{\smile})^{\smile} \\
\equiv^*_{(RA12)} & \top/(./R^{\smile})^{\smile} \\
\equiv^*_{(Pred6)} & \top/(.[R^{\smile}]/R^{\smile})^{\smile} \\
\equiv^*_{(RA2)} & \top/(.[R^{\smile}]/./R^{\smile})^{\smile} \\
\equiv^*_{(RA5,\ RA6,\ Pred8)} & \top/R/.[R^{\smile}] \\
\subseteq^*_{(Monotonicity)} & \top/.[R^{\smile}] \\
\equiv^*_{(Pred4,\ RA2)} & \top[R^{\smile}]
\end{array}$$

In the other direction,

$$\begin{array}{ll}
 & \top[R^{\smile}] \\
\equiv^*_{(RA6)} & \top[R^{\smile}]^{\smile\smile} \\
\equiv^*_{(Pred8)} & (.[R^{\smile}]/\top)^{\smile} \\
\equiv^*_{(Pred4)} & (./((R^{\smile}/\top)\ \texttt{intersect}\ .)/\top)^{\smile} \\
\equiv^*_{(RA5,\ RA12,\ RA14)} & \top/((R^{\smile}/\top)\ \texttt{intersect}\ .)^{\smile}/. \\
\equiv^*_{(RA2)} & \top/((R^{\smile}/\top)\ \texttt{intersect}\ .)^{\smile} \\
\equiv^*_{(RA5,\ RA6,\ RA14)} & \top/((\top/R)\ \texttt{intersect}\ .) \\
\subseteq^*_{(Boolean,\ Monotonicity)} & \top/\top/R \\
\equiv^*_{RA11} & \top/R
\end{array}$$

Now derive *Pred9*:

$$\begin{array}{ll}
 & \top[R^{\smile}/S] \\
\equiv^*_{(Pred7)} & \top[R^{\smile}[S]] \\
\equiv^*_{(RA6)} & \top[(R^{\smile}[S])^{\smile\smile}] \\
\equiv^*_{(\top/R \equiv^* \top[R^{\smile}])} & \top/(R^{\smile}[S])^{\smile} \\
\equiv^*_{(Pred8,\ RA6)} & \top/.[S]/R \\
\equiv^*_{(Pred5)} & \top[S]/R
\end{array}$$

▶ *Pred10*.

$$\begin{array}{ll}
 & R[S\ \texttt{and}\ \texttt{not}(T)] \\
\equiv^*_{(Pred1)} & R[S][\texttt{not}(T)] \\
\equiv^*_{(Pred3)} & R[S]\ \texttt{except}\ R[T] \\
\equiv^*_{(Boolean)} & R[(S\ \texttt{intersect}\ T)\ \texttt{union}\ (S\ \texttt{except}\ T)] \\
 & \quad \texttt{except}\ R[T] \\
\subseteq^*_{(Monotonicity)} & R[(S\ \texttt{intersect}\ T)\ \texttt{union}\ (S\ \texttt{except}\ T)] \\
 & \quad \texttt{except}\ R[(S\ \texttt{intersect}\ T)] \\
\equiv^*_{(Pred4,\ RA3,\ BA6,\ RA8)} & \big(R[(S\ \texttt{intersect}\ T)]\ \texttt{union}\ R[(S\ \texttt{except}\ T)]\big) \\
 & \quad \texttt{except}\ R[(S\ \texttt{intersect}\ T)] \\
\subseteq^*_{(Boolean)} & R[(S\ \texttt{except}\ T)]
\end{array}$$

▶ *Pred11*. In one direction,

$$\begin{array}{ll} & \top[\top[R]] \\ \equiv^*_{(Pred7)} & \top[\top/R] \\ \equiv^*_{(Pred4)} & \top/((\top/R/\top)\cap .) \\ \subseteq_{(Monotonicity,\ RA11)} & \top/R/\top \end{array}$$

In the other direction,

$$\begin{array}{ll} & \top/R/\top \\ \equiv^*_{(Pred6)} & \top[R]/R/\top \\ \equiv^*_{(Pred9)} & \top[R^{\smile}/R]/\top \\ \subseteq^*_{(Monotonicity)} & \top[\top/R]/\top \\ \equiv^*_{(Pred9,\ RA14,\ RA11)} & \top[\top/R] \\ \equiv^*_{(Pred7)} & \top[\top[R]] \end{array}$$ □

To simplify things, we will restrict attention to a subset of the language. We call a variable free path expression *normalized* if it contains no predicates (i.e., $\cdot$ [$\cdot$]) and it contains only steps `Axis::NameTest` for which `Axis` is `self` or `NameTest` is `*`. The following lemma shows that we can safely restrict attention to normalized path expressions.

Lemma 11 *Every variable free path expression R is provably equivalent to a variable free normalized path expression.*

Proof Predicates and steps of the form `Axis::NameTest` with `Axis` $\neq$ `self` and `NameTest` $\neq$ `*` can be eliminated by repeatedly applying the axioms *Pred1–Pred4* and *Tag1*. □

Restricting attention to normalized variable free path expressions has the advantage that we are in a purely relation algebraic setting (normalized path expressions are just expressions of Tarski's Relation Algebra, built from the atomic relations $\leftarrow, \rightarrow, \uparrow, \downarrow, \leftarrow^+, \rightarrow^+, \uparrow^+, \downarrow^+$, and `self::QName`) and hence the definitions and results from Sect. 3.2 can be sensibly applied. Note that restricting ourselves to normalized expressions in the statements of our Lemmas and Theorems does not preclude us from using expressions with predicates inside the proofs.

We now proceed to prove completeness. We build on Venema's results for Tarski's relation algebra that were described in Sect. 3.2. As a first step, we show that the theory generated by the axioms in Fig. 7 *provides Henkin witnesses*. For any path expression R, let us use last-pair(R) as a shorthand for R `except` ($\ll$ $/R/\top$ `union` $R/\gg$), which denotes *the subrelation of R consisting only of the last pair in lexicographic document order*. Also, let us use diff as a shorthand for ($\top$ `except` .), which defines the inequality relation.

Lemma 12 *The axiomatization in Fig. 7 provides Henkin witnesses. More specifically, for each normalized variable free path expression R,*

1. $\textsf{last-pair}(R)$ *is a nominal*, *i.e.*,

$$\textsf{last-pair}(R) \subseteq^* \top \ \texttt{except} \ ((\top/\textsf{last-pair}(R)/\textsf{diff}) \ \texttt{union} \ (\textsf{diff}/\textsf{last-pair}(R)/\top))$$

2. $\textsf{last-pair}(R) \subseteq^* R$
3. $\top/R/\top \equiv^* \top/\textsf{last-pair}(R)/\top$.

Proof

1. Consider

$$\textsf{diff}/\textsf{last-pair}(R)/\top \ \texttt{union} \ \top/\textsf{last-pair}(R)/\textsf{diff}.$$

Repeated application of *Tr11* (according to which $\top \equiv^* \texttt{.union} \ll \texttt{union} \gg$), *RA3*, *RA8* and *Monotonicity* shows that this expression is provably equivalent to the union of the following four expressions:
(a) $\ll /\textsf{last-pair}(R)/\top$
(b) $\gg /\textsf{last-pair}(R)/\top$
(c) $./\textsf{last-pair}(R)/\ll$
(d) $./\textsf{last-pair}(R)/\gg$.
Each of these expressions is provably contained in $(\top \ \texttt{except} \ \textsf{last-pair}(R))$:
(a)

$$\begin{array}{ll} & \ll /\textsf{last-pair}(R)/\top \\ \subseteq^*_{\textit{(Monotonicity)}} & \ll /R/\top \\ \subseteq^*_{\textit{(Boolean)}} & \top \ \texttt{except} \ \textsf{last-pair}(R) \end{array}$$

(b)

$$\begin{array}{ll} & \gg /\textsf{last-pair}(R)/\top \\ \subseteq^*_{\textit{(Monotonicity)}} & \gg /(\top \ \texttt{except} \ (\ll /R/\top))/\top \\ \subseteq^*_{\textit{(RA7)}} & (\top \ \texttt{except} \ R/\top)/\top \\ \subseteq^*_{\textit{(RA14)}} & (\top \ \texttt{except} \ R/\top)/\top^{\smile} \\ \subseteq^*_{\textit{(RA17)}} & \top \ \texttt{except} \ R \\ \subseteq^*_{\textit{(Monotonicity)}} & \top \ \texttt{except} \ \textsf{last-pair}(R) \end{array}$$

(c)

$$\begin{array}{ll} & ./\textsf{last-pair}(R)/\ll \\ \equiv^*_{(RA9)} & \textsf{last-pair}(R)/\ll \\ \subseteq^*_{\textit{(Monotonicity)}} & (\top \ \texttt{except} \ R/\gg)/\ll \\ \subseteq^*_{\textit{(RA17)}} & \top \ \texttt{except} \ R \\ \subseteq^*_{\textit{(Monotonicity)}} & \top \ \texttt{except} \ \textsf{last-pair}(R) \end{array}$$

(d)

$$\begin{array}{ll} & ./\textsf{last-pair}(R)/\gg \\ \equiv^*_{\textit{(RA9)}} & \textsf{last-pair}(R)/\gg \\ \subseteq^*_{\textit{(Monotonicity)}} & R/\gg \\ \subseteq^*_{\textit{(Boolean)}} & \top \ \texttt{except} \ \textsf{last-pair}(R) \end{array}$$

It follows by Boolean reasoning that

$$\textsf{last-pair}(R) \subseteq^* \top \texttt{ except } ((\top/\textsf{last-pair}(R)/\textsf{diff}) \texttt{ union } (\textsf{diff}/\textsf{last-pair}(R)/\top))$$

2. By Boolean reasoning.
3. By *(Pred11)*, it suffices to show that, for any expression R, $\top[\top[R]] \equiv^* \top[\top[\textsf{last-pair}(R)]]$. In one direction, containment follows from the monotonicity lemma. For containment in the other direction, the derivation is as follows:

$$\begin{array}{ll}
 & \top[\top[R]] \\
\equiv^*_{\textit{(Ind, Pred5)}} & \top[\top[R \texttt{ except } R/\gg]] \\
\equiv^*_{\textit{(Ind, Pred5)}} & \top[\top[R \texttt{ except } R/\gg] \texttt{ except} \\
 & \top[R \texttt{ except } R/\gg]/\gg] \\
\equiv^*_{\textit{(Pred7, Pred9)}} & \top[\top[R \texttt{ except } R/\gg] \texttt{ except} \\
 & \top[\ll[(R \texttt{ except } R/\gg)]]] \\
\equiv^*_{\textit{(Ind, Pred5)}} & \top[\top[R \texttt{ except } R/\gg] \texttt{ except } \top[\ll[R]]] \\
\equiv^*_{\textit{(Pred4, RA11)}} & \top[\top[R \texttt{ except } R/\gg] \texttt{ except } \top[\ll[R/\top]]] \\
\equiv^*_{\textit{(Pred7)}} & \top[\top[R \texttt{ except } R/\gg] \texttt{ except } \top[\ll/R/\top]] \\
\subseteq^*_{\textit{(Monotonicity)}} & \top[\top[R \texttt{ except } R/\gg] \texttt{ except} \\
 & \top[R \texttt{ except } R/\gg][\ll/R/\top]] \\
\equiv^*_{\textit{(Pred3)}} & \top[\top[(R \texttt{ except } R/\gg)][\texttt{not}(\ll/R/\top)]] \\
\equiv^*_{\textit{(Pred1)}} & \top[\top[(R \texttt{ except } R/\gg) \texttt{ and not}(\ll/R/\top)]] \\
\subseteq^*_{\textit{(Pred10)}} & \top[\top[(R \texttt{ except } R/\gg) \texttt{ except } (\ll/R/\top)]] \\
\equiv^*_{\textit{(Boolean)}} & \top[\top[\textsf{last-pair}(R)]]
\end{array}$$

□

We are now ready to proceed with the proof of Theorem 9.

Proof of Theorem 9 The proof essentially proceeds in two steps: first, we will prove completeness with respect to arbitrary structures (i.e., possibly infinite, and in which the atomic expressions $\downarrow^+$, $\rightarrow^+$, etc. can denote arbitrary binary relations) satisfying all instances of the axioms, then we prove completeness with respect to actual finite trees.

Let Σ be the collection of all instances of the axiom schemes in Fig. 7.

In what follows, we will restrict attention to normalized path expressions. This is safe by Lemma 11. □

Claim 1 *For all normalized variable free path expressions R, S, if $R \not\equiv^* S$ then there is a structure (not necessarily a finite tree) satisfying Σ, and a pair (d, e) of elements of the domain of the structure, such that (d, e) stands in the relation $(R \texttt{ except } S) \texttt{ union } (S \texttt{ except } R)$.*

Proof of Claim This follows from Lemma 8, by the fact that our axiomatization provides Henkin witnesses, cf. Lemma 12. Note that, by Lemma 10, the missing *RA* axioms *RA4–RA6* are derivable. □

Axioms enforcing that each variable denotes a constant function:

Var1.	$\top/\$i$	$\equiv$	$\$i$
Var2.	$\$i/(\top$ `except` $.)$	$\equiv$	$\top$ `except` $\$i$

Axiom for node-equality tests:

Eq1.	$R[S$ is $T]$	$\equiv$	$R[S$ `intersect` $T]$

Rule for eliminating variables in a derivation:

Name. From $\$i/R \equiv \$i/S$ derive $R \equiv S$, provided $\$i$ does not occur in R, S

Axioms for `for`:

For1.	$\$x/$`for` $\$y$ `in` R `return` S	$\equiv$	$\$x/R/($`for` $\$y$ `in` $.$ `return` $\$x/S)$
For2.	$\$x/$`for` $\$y$ `in` $.$ `return` S	$\equiv$	$\$x/S^{\$y}_{\$x}$

where $S^{\$y}_{\$x}$ is the result of replacing all free occurrences of $\$y$ by $\$x$, provided that this is a safe substitution.

Fig. 9 Additional axioms for variables and `for`

Claim 2 *Let $\mathfrak{M} = (M, (R_a)_{a\in\{\downarrow,\uparrow,\leftarrow,\rightarrow,\downarrow^+,\uparrow^+,\leftarrow^+,\rightarrow^+\}}, (R_{\mathtt{self::}p})_{p\in\mathtt{QName}})$ be any structure satisfying Σ. Then* (i) *the relations $(R_a)_{a\in\{\downarrow,\uparrow,\leftarrow,\rightarrow,\downarrow^+,\uparrow^+,\leftarrow^+,\rightarrow^+\}}$ are all definable in terms of the relations $R_{\downarrow^+}$ and $R_{\rightarrow^+}$, in the natural way, and $(R_{\mathtt{self::}p})_{p\in\mathtt{QName}}$ are subrelations of the identity relation over M, and* (ii) *the structure $(M, R_{\downarrow^+}, R_{\rightarrow^+}, (P_p)_{p\in\mathtt{QName}})$ satisfies all FO axioms in Fig. 5, where $P_p = \{d \in M \mid (d, d) \in R_p\}$ for all $p \in$* QName.

Proof of Claim The first half of the claim follows from the axioms Tr3b, Tr8b, the derivable RA6, and axiom Tag2. For the second half of the claim, since the axioms *Tr1–Tr11* all hold in $\mathfrak{M}$, $\mathfrak{M}$ satisfies the first-order properties defined by these axioms, which imply the first-order formulas *QT1–QT11* of Fig. 5. For instance, the axioms *QT1* and *Tr1* both express that $R_{\downarrow^+}$ is transitive. Similarly for the other axioms. Similarly, one can show that all instances of the *QT-Ind* axiom hold in M: let $\phi(x)$ be any FO formula in one free variable. By Theorem 5, there is a variable free Core XPath 2.0 PathExpr R such that in any model of *QT1–QT11*, R denotes the set of all pairs (x, y) with y satisfying ϕ. We know (taking the appropriate instance of the Ind axiom of Fig. 6) that $.[R]$ and $.[R$ `except` $R/\gg]$ denote the same relation in our model. But this means that if there is any node satisfying $\phi(x)$, then there is a last one in document-order. This is precisely what the QT-Ind axiom of Fig. 4 says (cf. footnote 4). □

It follows by Theorem 4 that every first-order formula, and therefore also every Core XPath 2.0 expression, satisfied in M is satisfied in some finite tree. In particular, this holds for $(R$ `except` $S)$ `union` $(S$ `except` $R)$.

4.2 Adding Variables

We will now extend our axiomatization to the full language of Core XPath 2.0 with variables and `for`-expressions, by adding a number of extra axioms, and an extra rule of inference, listed in Fig. 9. Naturally, we now take the symbols $R, S, \ldots$ and $X, Y, \ldots$ in Figs. 7 and 9 to range over path expressions and test expressions, respectively, that are *not necessarily variable free*.

The *For2* axiom deserves special attention, as it involves variable substitution. As is customary for quantified logics such as first-order logic, we prohibit unsafe substitution. For example, replacing the free occurrence of $\$j$ by $\$i$ in $\texttt{for \$}i\ \texttt{in}\ \downarrow\ \texttt{return \$}i\texttt{[not(. is \$}j\texttt{)]}$ constitutes an *unsafe* substitution, because it has the side-effect that the variable becomes bound by the `for`-quantifier.[5]

The inference rule *Name* allows us, when trying to prove equivalence of two expressions, to assign a variable to the current node. This sometimes makes it easier to prove the equivalence.

At this point, it is worth discussing in some detail a distinction between two types of axiomatizations. An *orthodox* axiomatization is one that uses only the standard rules of inference from equational logic, i.e., reflexivity, transitivity and symmetry of the equality relation, and the rule for replacement of equals by equals. A *non-orthodox* axiomatization may use additional rules of inference. In particular, the axiomatization for the variable free fragment of Core XPath 2.0 that we gave in the previous section was an orthodox axiomatization, while the axiomatization for the full Core XPath 2.0 language that we introduce in the present section is non-orthodox, as it involves the new inference rule *Name*. In general, orthodox axiomatizations are preferable, as derivations in them can be represented as sequences of rewriting steps starting with one expression and ending with another, where each rewriting step corresponds to replacing a sub-expression by another, equivalent, one. Unfortunately, we have not been able to find an orthodox axiomatization for the full Core XPath 2.0 language. Still, we believe that the non-orthodox axiomatization we present here is of value, because both the axioms and the inference rule are reasonably natural (unlike an axiomatization that would be obtained, say, by taking a standard complete set of axioms and inference rules for first-order logic and translating them directly into Core XPath 2.0 equations).

We now proceed with the proof of completeness.

Lemma 13 (A variant of the Name rule) *For all path expressions R and S, and for all variables $\$i$ not occurring in R and S, if* $(\$i/R/\top\ \texttt{union}\ (\top\ \texttt{except}\ (\top/R/\top))$ $\subseteq^* S$ *then* $S \equiv^* \top$.

Proof Before we commence with the proof, let us first show that $\$i/\$i \equiv^* \$i$. In one direction, $\$i/\$i \subseteq^*_{(Monotonicity)} \top/\$i \equiv^*_{(Var1)} \$i$. In the other direction,

$$\begin{array}{ll}
 & \$i \\
\equiv^*_{(Var1)} & \top/\$i \\
\equiv^*_{(Boolean,\ RA3)} & \$i/\$i\ \texttt{union}\ (\top\ \texttt{except}\ \$i)/\$i \\
\equiv^*_{(Var2)} & \$i/\$i\ \texttt{union}\ \$i/(\top\ \texttt{except}\ \$i)/\$i \\
\subseteq^*_{(Monotonicity)} & \$i/\$i\ \texttt{union}\ \$i/\top/\$i \\
\equiv^*_{(Var1)} & \$i/\$i\ \texttt{union}\ \$i/\$i \\
\equiv^*_{(Boolean)} & \$i/\$i
\end{array}$$

[5] Formally, substituting $\$y$ by $\$x$ in R is safe if no free occurrence of $\$y$ in R is in the scope of a `for` $\$x$ operator, where an occurrence of $\$y$ is said to be free if it is not in the scope of a `for` $\$y$ operator. If a substitution is not safe, one can always make it safe by renaming the bound variables in the expression.

Now for the main proof. Assume $(\$i/R/\top \texttt{ union } (\top \texttt{ except } (\top/R/\top)) \subseteq^* S$. We will show that both $R/\top \subseteq^* S$ and $\top \texttt{ except } (R/\top) \subseteq^* S$, and therefore, by Boolean reasoning, $S \equiv^* \top$.

First, note that, by Boolean reasoning,

(1) $\$i/R/\top \subseteq^* S$ and
(2) $(\top \texttt{ except } (\top/R/\top)) \subseteq^* S$.

From (1), we obtain that $\$i/R/\top \subseteq^* \$i/\$i/R/\top \subseteq^*_{(Monotonicity)} \$i/S$, and therefore, by the Name rule, $(R/\top) \subseteq^* S$.

From (2), we obtain that $(\top \texttt{ except } (R/\top)) \subseteq^* S$, by the following derivation:

$$\begin{array}{ll} & (\top \texttt{ except } (R/\top)) \\ \equiv^*_{(RA9)} & ./(\top \texttt{ except } (R/\top)) \\ \subseteq^*_{(Monotonicity)} & \top/(\top \texttt{ except } (R/\top)) \\ \subseteq^*_{(RA7,RA14)} & S \end{array}$$ □

Theorem 14 (Completeness) *For all equivalent expressions R, S, $R \equiv^* S$.*

Proof The completeness proof proceeds as before, with two important differences: (i) the language now contains variables and for-clauses. However, most of this proof, we will treat these simply as atomic expressions (thus we treat expressions of the form $\texttt{for } \$i \texttt{ in } R \texttt{ return } S$ as atomic, ignoring their internal structure). (ii) We prove a variant of Lemma 7.

Claim 3 *If $R \not\equiv^* S$, then there is a set Σ of equivalences, such that $\Sigma \not\models R \equiv S$, and such that for every path expression T there is a variable $\$i$ such that $\top/T/\top \equiv \$i/T/\top \in \Sigma$.*

Proof of Claim Let T be short for $\top/((R \texttt{ except } S) \cup (S \texttt{ except } R))/\top$. Let $R_1, R_2, \ldots$ be an enumeration of all (countably many) expressions of the language. For each R_k ($k \in \mathbb{N}$) pick a variable $\$i_k$ distinct from $\$i_1, \ldots, \$i_{k-1}$ and not occurring in $R_1, \ldots, R_k$. (This can always be done with a countably infinite supply of variables.) For $\ell \geq 0$, let

$$\begin{aligned} \Sigma_\ell = \{T \equiv \top\} \cup \{(\$i_k/R_k/\top) \texttt{ union } \\ (\top \texttt{ except } \top/R_k/\top) \mid k \leq \ell\}. \end{aligned}$$

We claim that each Σ_ℓ is consistent (i.e., is such that no intersection of finitely many expressions in the set implies $\bot$). The proof is by induction on ℓ. For $\ell = 0$, $\Sigma_\ell = \{S\}$, which is consistent by assumption. Next, suppose, for the sake of contradiction, that Σ_ℓ is consistent but $\Sigma_{\ell+1}$ is not. Then it follows by Boolean reasoning that $((\$i_{\ell+1}/R_{\ell+1}/\top) \texttt{ union } (\top \texttt{ except } \top/R_{\ell+1}/\top)) \equiv^* (\top \texttt{ except } \cap \Sigma_\ell)$. By Lemma 13, this implies that $\cap\Sigma_\ell \equiv^* \bot$, a contradiction.

It follows that $\Sigma = \bigcup_\ell \Sigma_\ell$ is also consistent, and by construction it satisfies all requirements. □

Next, we proceed as before: we construct a model M satisfying all instances of the axioms and satisfying all equivalences in Σ but not satisfying the equivalence $R \equiv S$.

Recall that we treat variables and for-clauses as atomic expressions, interpreted by the model. Now, the axioms *Var1* and *Var2* ensure that each variable denotes a constant function. As we will see next, the axioms *For1* and *For2* also force *for-clauses* to have the correct denotation conform Fig. 2. A special feature of Venema's [24] model construction is that every element d of the domain of the constructed model M is uniquely identified by some expression T, in the sense that $[\![T]\!]^M = \{(d,d)\}$. It follows by the construction of the set Σ that d is also uniquely identified by a variable $\$i$, in the sense that $rng([\$i]) = \{d\}$. It follows by the axioms *For1* and *For2* that for-clauses are correctly interpreted in M.

Finally, we use again Theorem 4 to conclude that every first-order formula, and therefore also every Core XPath 2.0 expression, satisfied in M is satisfied in some finite tree. In particular, this holds for (R `except` S) `union` (S `except` R). □

4.3 Further Remarks on the Axiomatization

Minimality Our axiomatization was not intended to be a minimal list of axioms. Rather than being concise, our aim was to formulate the axioms as naturally as possible. It is quite likely that some of the axioms can be derived from others.

Stronger Forms of Completeness Properly speaking, many of our axioms should be called *axiom schemes*. For instance, *BA1* says that R `union` (S `union` T) is equivalent to (R `union` S) `union` T *for all path expressions* R, S *and* T. This leads to an interesting question: suppose we enrich the language by allowing "*path variables*" $R, S, \ldots$ as atomic variables, and let $\mathcal{R}$ and $\mathcal{S}$ be two path expressions containing such path variables, such that $\mathcal{R}$ is equivalent to $\mathcal{S}$ *under all substitutions of path expressions for the path variables*. Then can we derive $\mathcal{R} \equiv \mathcal{S}$ as a scheme in our axiomatization? The answer is negative. In fact, no recursive axiomatization of Core XPath 2.0 can be complete in this strong sense, for the following reason.

Theorem 15 *Equivalence of Core XPath* 2.0 *path expressions containing a single path variable is undecidable.*

Proof (Sketch) It follows from results in [7, 20] that the equivalence problem for expressions of Tarski's relation algebra in a single binary relation, over finite models is undecidable. In terms of XPath, this means that equivalence of Core XPath 2.0 path expressions containing a variable standing for an arbitrary binary relation is undecidable. Since, within any given XML document, all binary relations are definable in Core XPath 2.0, this problem coincides with the equivalence problem for Core XPath 2.0 path expressions containing a variable standing for a Core XPath 2.0 path expression. □

Sequence Semantics Versus Set Semantics The soundness of some of our axioms (in particular, *RA2* and *Pred2*–*Pred4*) depends on the set-theoretic semantics we provided for Core XPath 2.0. If, as in the official XPath 2.0 semantics, expressions are taken to manipulate *sequences* rather than sets of nodes, these axioms no longer hold. However, our results are still meaningful in this setting.

First, observe that, if $\equiv$ is interpreted as *equivalence up to sorting and duplicate removal*, henceforth $\equiv_{sdr}$, then all axioms remain sound. Secondly, all equivalences *derivable* from the axioms remain sound as well, because $\equiv_{srd}$ is an equivalence relation, and it admits *replacement of equals by equals* (e.g., if $R \equiv_{sdr} R'$ and $S \equiv_{sdr} S'$ then `for $i in` R `return` $S \equiv_{sdr}$ `for $i in` R' `return` S'). Thus all derivable equivalences hold if equivalence is interpreted as equivalence up to sorting and duplicate removal. In fact, it is not hard to see that both axiomatizations (the one from Sect. 4.1 and the one from Sect. 4.2) are complete under this interpretation. In particular, for any path expression R, $R \equiv^* \perp$ iff R is unsatisfiable according to the sequence semantics.

5 Conclusions

We have defined the navigational core of XPath 2.0, analogously to Core XPath, and we gave an intuitive and rather simple axiomatization of the equivalence of relative path expressions by combining three sets of algebraic axioms: Tarski's axiom system RA of relation algebras, a first order axiomatization of the theory of finite ordered trees, and an axiomatization of the variable binder.

We have provided axiomatizations for the complete Core XPath 2.0 language and its equally expressive fragment without the for-loop and variables. By the linear embedding of first order logic, Core XPath 2.0 inherits its non-elementary space lower bound for query equivalence. This holds even for the variable-free fragment [23]. Still, we hope that our axiomatizations will be useful for query optimization. The connection with Tarski's relation algebras might be fruitfully exploited by using results from the relation algebra theorem proving community [5].

Of particular practical importance seems to be the elimination of the for-loop and the use of variables. Empirical tests have shown us that rewritings into the for-free fragment can lead to speedups of up to 3 orders of magnitude with commercial XQuery processors. We obtained these results with the following two queries:

(1) `descendant :: * [P] except(descendant :: *[not Q]/descendant :: *)`
(2) `for $start in . return descendant :: * [P and (#)]`,
with (#) the test expression
`every $i in ancestor :: * [ancestor :: * [. is $start]] satisfies $i[Q]`

where `P` and `Q` are filter expressions of the form `@attribute='value'`. Both express the relation `(child :: * [Q])`*`/child :: * [P]`, with $(\cdot)^*$ the transitive closure operation. By completeness, our axiom system can derive their equivalence. An important issue is whether such query rewritings can be performed efficiently.

Appendix: The First-Order Theory of Finite Trees

In this appendix, we prove Theorem 4 from Sect. 3, which we repeat here.

Theorem 4 *The FO theory of finite node-labeled sibling ordered trees is completely axiomatized by the axioms and axiom scheme in Fig.* 5.

One cannot hope to *define* the class of finite trees with first-order formulas. A simple compactness argument shows for instance that all first-order formulas valid on finite trees have infinite, and in fact non-wellfounded, models. Instead, the FO theory we will present defines a slightly larger class of structures, which we will call *definably wellfounded quasi-trees*. This will suffice, as we will show that a FO formula is valid on such structures iff it is valid on "real" finite trees.

We will consider structures of the form $T = (N, <, \prec, V)$, where N is intended to be the domain of a tree, $<$ and $\prec$ the (transitive) descendant and following-sibling relations, and V a function assigning subsets of N to the node labels. Recall that $x < y$ is intended to mean that y is a descendant of x.

Definition 16 A *quasi-tree* is any structure $(T, <, \prec, V)$ of the form described above, satisfying the axioms QT1–QT11 of Fig. 5. A quasi-tree is *definably wellfounded* if, in addition, it satisfies all instances of the induction scheme QT-Ind of Fig. 5 (where $\phi(x)$ ranges over FO formulas in one free variable).

It is not hard to see that every *finite* quasi-tree is indeed a finite tree. In the proof of the following result, we use a technique developed by Kees Doets in his PhD thesis [9]. Call two structures *n-equivalent*, for $n \in \mathbb{N}$, if they cannot be distinguished by a FO formula of quantifier depth n.

Lemma 17 *For all $n \in \mathbb{N}$, every definably wellfounded quasi-tree of finite signature is n-equivalent to a finite tree. In particular, a FO formula is valid on definably wellfounded quasi-trees iff it is valid on finite trees.*

Proof During this proof, it will be convenient to work with quasi-forests, i.e., ordered sequences of quasi-trees (for instance, such as obtained by removing the root from a quasi-tree). For any node a of a quasi-tree T, let T_a be the quasi-forest consisting of all the subtrees of T whose root is either a or a sibling to the right (an immediate following-sibling) of a. Now, let X_n be the set of all nodes a of T for which it holds that T_a is n-equivalent to a finite quasi-forest.

Claim 4 *X_n is invariant for $n+1$-equivalence (i.e., $(T, a) \equiv_{n+1} (T, b)$ implies that $a \in X_n$ iff $b \in X_n$), and hence is defined by a first-order formula of quantifier depth $n+1$.*

Proof of Claim Suppose that $(T, a) \equiv_{n+1} (T, b)$. We will show that $T_a \equiv_n T_b$, and hence, by the definition of X_n, $a \in X_n$ iff $b \in X_n$. Let ϕ be any formula of quantifier depth n, and let $\phi'(x)$ be the 'relativization' of ϕ obtained by replacing each existential quantifier $\exists y$ by $\exists y(\exists z(R^*_{\rightarrow}xz \wedge R^*_{\downarrow}zy) \wedge \cdots)$ and replacing each universal quantifier $\forall y$ by $\forall y(\exists z(R^*_\rightarrow xz \wedge R^*_\downarrow zy) \rightarrow \cdots)$. Notice that $\phi'(x)$ expresses precisely that ϕ holds within the subforest T_x. Moreover, the quantifier depth of ϕ is at most $n+1$. It follows that $(T,a) \models \phi'(x)$ iff $(T,b) \models \phi'(x)$, and hence $T_a \models \phi$ iff $T_b \models \phi$.

For the second part of the claim, note that there are only finitely many first-order formulas of any given quantifier depth, as the vocabulary is finite. □

Claim 5 *If all nodes following a in document order are in X_n, then a itself belongs to X_n.*

Proof of Claim Let us consider the case where a has a descendant and a following sibling (all other cases are simpler). Then, by axioms QT3, QT5, QT8, QT9 and QT10, a has a first child b, and an immediate following sibling c. Moreover, we know that both b and c are in X_n. In other words, T_b and T_c are n-equivalent to finite quasi-forests T'_b and T'_c. Now, we construct a finite quasi-forest T'_a by taking the disjoint union of T'_b and T'_c and adding one more point, which is a common parent of all roots of T'_b and a left sibling of all roots of T'_c. It is not hard to see that T'_a is again a finite quasi-forest. Moreover, $T_a \equiv_n T'_a$, as can easily be shown by means of Ehrenfeucht-Fraïssé games (the winning strategies of Duplicator for the n-round EF games between T_b and T'_b and between T_c and T'_c naturally combine into a winning strategy for the game between T_a and T'_a). □

It follows from these two claims, by the induction scheme for definable properties, that X_n contains all nodes of the tree, including the root, and hence T is n-equivalent to a finite quasi-tree. As we already noted earlier, every finite quasi-tree is in fact a finite tree. For the second statement of the lemma, it suffices to note that every FO formula has a finite vocabulary and a finite quantifier depth. □

Theorem 4 follows directly from this lemma.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Backofen, R., Rogers, J., Vijay-Shanker, K.: A first-order axiomatization of the theory of finite trees. J. Logic Lang. Inf. **4**(1), 5–39 (1995)
3. Benedikt, M., Fan, W., Geerts, F.: XPath satisfiability in the presence of DTDs. J. Assoc. Comput. Mach. **55**(2), (2008)
4. Benedikt, M., Fan, W., Kuper, G.: Structural properties of XPath fragments. Theor. Comput. Sci. **336**(1), 3–31 (2005)
5. Berghammer, R., Schmidt, G., Winter, M.: RelView and Rath—two systems for dealing with relations. In: Theory and Applications of Relational Structures as Knowledge Instruments. LNCS, vol. 2929, pp. 1–16. Springer, Berlin (2003)
6. Bojanczyk, M., Muscholl, A., Schwentick, Th., Segoufin, L., David, C.: Two-variable logic on words with data. In: Proceedings LICS'06, pp. 7–16 (2006)
7. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Springer, Berlin (1997)
8. Dawar, A.: How many first-order variables are needed on finite ordered structures? In: Artemov, S., et al. (eds.) We will Show Them: Essays in Honour of Dov Gabbay, pp. 489–520. College Publications, Woodend Vic. (2005)
9. Doets, H.C.: Completeness and definability: applications of the ehrenfeucht game in intensional and second-order logic. Ph.D. Thesis, Department of Mathematics and Computer Science, University of Amsterdam (1987)

10. Geerts, F., Fan, W.: Satisfiability of XPath queries with sibling axes. In: Proceedings DBPL'05, pp. 122–137 (2005)
11. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. In: VLDB'02, pp. 95–106 (2002)
12. Gyssens, M., Paredaens, J., Van Gucht, D., Fletcher, G.: Structural characterizations of the semantics of XPath as navigation tool on a document. In: Proceedings PODS'06, pp. 318–327 (2006)
13. Henkin, L., Monk, J.D., Tarski, A.: Cylindric Algebras, Part II. North-Holland, Amsterdam (1985)
14. Hidders, J.: Satisfiability of XPath expressions. In: Proceedings DBPL. LNCS, vol. 2921, pp. 21–36. Springer, Berlin (2003)
15. Hirsch, R., Hodkinson, I.: Relation algebras by games. In: Studies in Logic and the Foundations of Mathematics, vol. 147. North-Holland, Amsterdam (2002)
16. Kay, M.: XPath 2.0 Programmer's Reference. Wrox (2004)
17. Lyndon, R.C.: The representation of relation algebras. Ann. Math. **51**, 707–729 (1950)
18. Lyndon, R.C.: The representation of relation algebras, Part II. Ann. Math. **63**, 294–307 (1956)
19. Marx, M.: Conditional XPath. ACM Trans. Database Syst. (TODS) **30**(4), 929–959 (2005)
20. Tarski, A.: On the calculus of relations. J. Symb. Log. **6**, 73–89 (1941)
21. Tarski, A., Givant, S.: A Formalization of Set Theory without Variables. AMS Colloquium Publications, vol. 41. Rhode Island, Providence (1987)
22. ten Cate, B., Litak, T., Marx, M.: Complete axiomatizations for XPath fragments. In: Proceedings LID (Logic in Databases) Rome, Italy, 19–20 May 2008
23. ten Cate, B., Lutz, C.: The complexity of query containment in expressive fragments of XPath 2.0. In: Proceedings PODS'07 (2007)
24. Venema, Y.: Many-dimensional modal logic. Ph.D. Thesis, Institute for Logic, Language and Computation, University of Amsterdam (1992)
25. Venema, Y.: Completeness by completeness: since and until. In: de Rijke, M. (ed.) Diamonds and Defaults, pp. 349–358. Kluwer Academic, Dordrecht (1993)
26. Venema, Y.: Derivation rules as anti-axioms in modal logic. J. Symb. Log. **58**(3), 1003–1034 (1993)
27. Venema, Y.: Completeness through flatness. In: Gabbay, D., Ohlbach, H.J. (eds.) Temporal Logic, First International Conference ICTL'94, pp. 149–164 (1994)
28. Venema, Y.: Cylindrical modal logic. J. Symb. Log. **60**(2), 591–623 (1995)