Navigational XPath: calculus and algebra

Balder ten Cate
ISLA – Informatics Institute
Universiteit van Amsterdam
balder.tencate@uva.nl

Maarten Marx
ISLA – Informatics Institute
Universiteit van Amsterdam
marx@science.uva.nl

ABSTRACT

We survey expressivity results for navigational fragments of XPath 1.0 and 2.0, as well as Regular XPath$^{\approx}$. We also investigate algebras for these fragments.

1. INTRODUCTION

XPath is a common fragment of the XML querying and processing languages XQuery and XSLT, used for navigation through XML documents. In this paper we address two foundational issues concerning this language: (1) its *expressivity* in comparison to the first-order logic, and (2) *algebras* for XPath.

We focus on the *navigational* part of XPath: the part that is concerned purely with document navigation, not considering operations involving strings, numbers, or any other types of atomic content. Several navigational fragments of XPath 1.0 and 2.0 have been proposed [10, 26]. All in all, we consider four navigational XPath dialects: Core XPath 1.0, variable-free Core XPath 2.0, Core XPath 2.0 with variables, and Regular XPath$^{\approx}$.

1.1 XML tree navigation using path expressions

Path expressions describe ways of navigating through XML documents, i.e., traveling from one node to another in the tree. This means we can model the meaning of a path expressions by a binary relation on the nodes of the tree. For example, the XPath 1.0 path expression `descendant::p` (abbreviated as `.//p`) denotes in any XML tree T, the set of all pairs (m, n) with n a descendant node of m that has tag name `p`. Of course, binary relations can be defined using many other formalisms, e.g., by means of a first-order formula in two free variables. In the case of this example, the binary relation is equivalently expressed by the conjunctive query

$$\phi(x, y) = \texttt{descendant}(x, y) \wedge \texttt{p}(y) \ .$$

Conversely, the conjunctive query

$$\phi(x, y) = \exists z_1 \dots z_n \bigwedge_{i=1}^{n} \texttt{descendant}(x, z_i) \wedge \texttt{p}_i(z_i) \wedge \texttt{descendant}(z_i, y) \wedge \texttt{q}(y) \tag{1}$$

defines a binary relation that can be defined in XPath 1.0 by the union of the path expressions

$$\texttt{descendant} :: \texttt{p}_{\rho(1)} / \cdots / \texttt{descendant} :: \texttt{p}_{\rho(\texttt{n})} / \texttt{descendant} :: \texttt{q}$$

for all, exponentially many, permutations ρ of $1 \dots n$.

Next, consider the following first-order binary relation (familiar from temporal logic, and raising children):

$$\phi(x, y) = \texttt{descendant}(x, y) \wedge \texttt{q}(y) \wedge \forall z(\texttt{descendant}(x, z) \wedge \texttt{descendant}(z, y) \rightarrow \texttt{p}(z)) \tag{2}$$

A pair (m, n) stands in this relation if n is a descendant of m with tag name q and all nodes in-between m and n in the tree have tag name p. Can we express this in XPath 1.0?[1]

Questions such as these are hard to answer for languages as rich as full XPath 1.0 (whose technical specification is about 30 pages long). In order to be able to give a mathematically precise answer, in [19] the same question was studied in the context of *Core XPath 1.0* [10]. This is a compact, well defined fragment of XPath 1.0 with a clean logical semantics. It captures the navigational core of XPath 1.0, abstracting away from operations involving strings, numbers, or any other types of atomic content. It was shown in [19] that (2) cannot be defined in Core XPath.

1.2 Ways of extending Core XPath 1.0

Various extensions of XPath 1.0 have been proposed, including the official W3C standard of XPath 2.0. With more expressive power, new binary relations can be defined and sometimes older ones can be defined more succinctly. We give examples of both, starting with the latter.

XPath 2.0 has an `intersect` operator: `Path1 intersect Path2` denotes the intersection of the binary

*** Database Principles Column.** Column editor: Leonid Libkin, School of Informatics, University of Edinburgh, Edinburgh, EH8 9LE, UK. E-mail: libkin@inf.ed.ac.uk.

[1] Note that `.//q[not(ancestor :: *[not(self :: p)])]` does not define the intended relation: it is only correct for pairs (m, n) where m is the root.

relations defined by Path1 and Path2. Using intersection, (1) can be expressed without exponential blow-up:

$$\begin{array}{l}\texttt{descendant} :: \texttt{p}_1/\texttt{descendant} :: \texttt{q} \ \ \texttt{intersect}\\ \texttt{descendant} :: \texttt{p}_2/\texttt{descendant} :: \texttt{q} \ \ \texttt{intersect}\\ \quad \ldots\\ \texttt{descendant} :: \texttt{p}_n/\texttt{descendant} :: \texttt{q}\end{array}$$

Similarly, the previously undefinable "until" relation (2) can be defined in various ways using additional operators that have been proposed. A first possibility is to use the Kleene star, inspired by [1]:

$$(\texttt{child} :: \texttt{p})^*/\texttt{child} :: \texttt{q}.$$

Here $(\texttt{Path})^*$ denotes the reflexive transitive closure of the binary relation denoted by Path. The Kleene star does not belong to XPath 1.0 or 2.0, but extensions of XPath with this operator have been proposed and implemented [24, 7, 6]. A second solution is to use the *path complementation* operator except that was introduced in XPath 2.0:

$$\begin{array}{l}\texttt{descendant} :: \texttt{q} \ \texttt{except}\\ \quad \texttt{descendant} :: *[\texttt{not}(\texttt{self} :: \texttt{p})]/\texttt{descendant} :: \texttt{q}\end{array}$$

Finally, a third option is to use quantified variables, which is possible in XPath 2.0 using the for-construct. Using for, we can write (2) as follows:

$$\begin{array}{l}\texttt{for \$s in . return}\\ \quad \texttt{descendant} :: \texttt{q}\big[\texttt{not}(\texttt{ancestor} :: *[\texttt{not}(\texttt{self} :: \texttt{p})]/\\ \qquad \texttt{ancestor} :: *[\texttt{. is \$s}])\big]\end{array}$$

Notice how the variable $s stores the initial node.

1.3 Two main questions of this paper

In this paper, we consider Core XPath 1.0 and three extensions of it, roughly corresponding to XPath 2.0, the variable free fragment of XPath 2.0, and an extension of XPath with transitive closure and path equalities. For each of these, we study two main questions: *what is the expressive power* and *what are suitable algebras*.

Expressivity and Codd completeness.

When a new query language is introduced, it is always useful to compare its expressive power to existing languages. E.F. Codd did this for SQL and relational algebra by showing that they are equally expressive as first-order logic [4]. With the navigational languages for XML we can do the same: given a dialect of XPath, we can ask how it compares to (fragments or extensions of) first-order logic. We explore this in Section 3.

Algebras for navigational XPath.

An important step towards efficient query evaluation is to identify a suitable algebra in which query plans can be formulated. Which algebra are suitable for our XPath dialects? In answering this question, we guide ourselves by the following criteria:

1. expressions in the XPath dialect should be efficiently translatable to algebraic expressions,
2. the algebra should not be much more expressive than the XPath dialect requires,
3. the algebra should not have much harder query evaluation or equivalence problem than the XPath dialect itself, and
4. there should be a nice set of algebraic equivalence rules for the algebra.

In Section 4, we will consider several candidates, such as Codd's relational algebra (CRA) and Tarski's algebra of binary relations (TRA). For each dialect of navigational XPath, a different algebra turns out to fit best.

Table 1: Syntax of Core XPath 1.0.

Axis	:=	self | child | parent | right | left
		| descendant
		| ancestor
		| following
		| preceding
		| following_sibling
		| preceding_sibling
NameTest	:=	QName | *
Step	:=	Axis::NameTest .
PathExpr	:=	Step
		| PathExpr/PathExpr
		| PathExpr union PathExpr
		| PathExpr[NodeExpr]
NodeExpr	:=	PathExpr
		| not NodeExpr
		| NodeExpr and NodeExpr
		| NodeExpr or NodeExpr.

2. PRELIMINARIES: FOUR DIALECTS OF NAVIGATIONAL XPATH

In this section, we review the syntax and semantics of Core XPath 1.0 —the navigational fragment of XPath 1.0 introduced in [10]— as well as three extensions.

Core XPath 1.0.

Core XPath 1.0 was introduced in [10] to capture the navigational core of XPath 1.0. The definition we will give here is from [18], which differs from the one of [10] as (1) it include the "one-step sibling axes" left, right (which are definable in XPath 1.0 using numerical predicates), (2) filters can be applied to any expression, and (3) we include the union operator on path expressions.

Table 1 gives the syntax of Core XPath 1.0. Here QName stands for any XML tag name. The primary type of expression is a *path expression* (PathExpr). Table 2 gives the semantics. Expressions are evaluated on finite sibling-ordered unranked trees whose nodes are labeled by XML tag names. Given such a tree,

Table 2: Semantics of Core XPath 1.0.

$$
\begin{array}{lcl}
[\![\mathsf{Axis} :: N]\!]_{\mathsf{PExpr}} & = & \{(x,y) \mid x\mathsf{Axis}y \text{ holds in the tree, and } y \text{ has tag } N\} \\
[\![\mathsf{Axis} :: *]\!]_{\mathsf{PExpr}} & = & \{(x,y) \mid x\mathsf{Axis}y \text{ holds in the tree}\} \\
[\![R/S]\!]_{\mathsf{PExpr}} & = & [\![R]\!]_{\mathsf{PExpr}} \circ [\![S]\!]_{\mathsf{PExpr}} \\
[\![R \texttt{ union } S]\!]_{\mathsf{PExpr}} & = & [\![R]\!]_{\mathsf{PExpr}} \cup [\![S]\!]_{\mathsf{PExpr}} \\
[\![R[T]]\!]_{\mathsf{PExpr}} & = & \{(x,y) \mid (x,y) \in [\![R]\!]_{\mathsf{PExpr}} \text{ and } y \in [\![\mathtt{T}]\!]_{\mathsf{NExpr}}\}
\end{array}
$$

$$
\begin{array}{lcl}
[\![\mathsf{PathExpr}]\!]_{\mathsf{NExpr}} & = & \{x \mid \exists y.(x,y) \in [\![\mathsf{PathExpr}]\!]_{\mathsf{PExpr}}\} \\
[\![\texttt{not } T]\!]_{\mathsf{NExpr}} & = & \{x \mid x \notin [\![T]\!]_{\mathsf{NExpr}}\} \\
[\![T_1 \texttt{ and } T_2]\!]_{\mathsf{NExpr}} & = & [\![T_1]\!]_{\mathsf{NExpr}} \cap [\![T_2]\!]_{\mathsf{NExpr}} \\
[\![T_1 \texttt{ or } T_2]\!]_{\mathsf{NExpr}} & = & [\![T_1]\!]_{\mathsf{NExpr}} \cup [\![T_2]\!]_{\mathsf{NExpr}}
\end{array}
$$

the meaning $[\![R]\!]_{\mathsf{PExpr}}$ of a PathExpr R is always a binary relation. This is just another, equivalent, way of specifying a function from nodes to sets of nodes (the answer-set semantics). The meaning $[\![T]\!]_{\mathsf{NExpr}}$ of a node expression T is always a set of nodes.

We will study the complexity of two tasks: query evaluation and query containment. For *query evaluation*, we will consider the *combined complexity* of the following problem: given a path expression, an XML-tree (suitably encoded) and a pair of nodes, determine whether the pair belongs to the relation denoted by the path expression. In the case of the *query containment* problem, the task is to determine, given two path expressions R, S, whether in every tree model, $[\![R]\!]_{\mathsf{PExpr}} \subseteq [\![S]\!]_{\mathsf{PExpr}}$. For Core XPath 1.0, the query evaluation problem for Core XPath 1.0 is in PTIME (in fact, it can be performed in linear time) [10], and the query containment problem is EXPTIME-complete [20, 17].

Core XPath 2.0 without variables.

In [26], Core XPath 2.0 was introduced as a navigational core of XPath 2.0 with a clean, logical semantics. One important simplifying assumption underlies Core XPath 2.0, namely that path expressions still denote *binary relations between nodes*, as they did in Core XPath 1.0. This is not the case in the full XPath 2.0, where they denote functions from nodes to sequences of nodes (not necessarily in document order and possibly containing duplicates). We follow the definition of Core XPath 2.0 from [26].

First, we consider the variable-free fragment of Core XPath 2.0. This is a very simple extension of Core XPath 1.0: it differs from Core XPath 1.0 only in that one can take intersections and complements of path expressions:

$$
\begin{array}{lcl}
[\![R \texttt{ intersect } S]\!]_{\mathsf{PExpr}} & = & [\![R]\!]_{\mathsf{PExpr}} \cap [\![S]\!]_{\mathsf{PExpr}} \\
[\![R \texttt{ except } S]\!]_{\mathsf{PExpr}} & = & [\![R]\!]_{\mathsf{PExpr}} \setminus [\![S]\!]_{\mathsf{PExpr}}.
\end{array}
$$

These operators do not only increase the expressive power of the language (as we will see in the next section), they also greatly increase its complexity. The query evaluation problem for variable free Core XPath 2.0 is still in PTIME (in fact, it can be performed in quadratic time), but the query containment problem is non-elementary (2-EXPTIME-complete for expressions without the complementation operator) [25].

Core XPath 2.0.

Besides the addition of the `intersect` and `except` operators, an important difference between XPath 1.0 and 2.0 is the use of quantified variables by means of the `for` construct. Formally, let a NodeRef expression be an expression of the form `$i` or . (where `$i` is a variable ranging over nodes in the tree). Then the syntax of full Core XPath 2.0 is obtained by extending the syntax of Core XPath 1.0 with the `intersect` and `except` operators from above, with path expressions of the form `$i` and `for $i in` PathExpr `return` PathExpr, and with node expressions of the form NodeRef `is` NodeRef. The latter tests whether the two expressions refer to the same node.

Since the expressions of Core XPath 2.0 can contain variables, the semantic interpretation is relative to an *assignment*, i.e., a function mapping variables to nodes. For g an assignment, `$i` a variable, and x a node, $g[\$\mathtt{i} \mapsto x]$ denotes the assignment g' which is identical to g except that $g'(i) = x$. Also, for any assignment g, node x, and NodeRef expression a, let $[\![a]\!]^{g,x}$ be $g(a)$ in case a is a variable, or x in case a is '.'. The semantics of the new constructs is as follows:

$$[\![\$\mathtt{i}]\!]^g_{\mathsf{PExpr}} = \{(x,y) \mid g(i) = y\}$$

$$[\![\texttt{for } \$\mathtt{i} \texttt{ in } R \texttt{ return } S]\!]^g_{\mathsf{PExpr}} = \{(x,y) \mid \exists z((x,z) \in [\![R]\!]^g_{\mathsf{PExpr}} \text{ and } (x,y) \in [\![S]\!]^{g[\$i \mapsto z]}_{\mathsf{PExpr}})\}$$

$$[\![a \texttt{ is } b]\!]_{\mathsf{NExpr}} = \{x \mid [\![a]\!]^{g,x} = [\![b]\!]^{g,x}\}.$$

The query evaluation problem for Core XPath 2.0 is PSPACE-complete, and the query containment problem is non-elementary [25].

Regular XPath$^{\approx}$.

Regular XPath$^{\approx}$ extends Core XPath 1.0 with two operators that are not part of XPath 1.0 or 2.0, and that, as we will see, make it more expressive. The most important of these is the Kleene star, which allows us to take the reflexive transitive closure of arbitrary path expressions. The other is *path equalities* (not to be confused with data value equalities). Formally, the semantics of these operators is as follows [24]:

$$[\![R^*]\!]_{\mathsf{PExpr}} = \text{reflexive transitive closure of } [\![R]\!]_{\mathsf{PExpr}}$$

$$[\![R \approx S]\!]_{\mathsf{NExpr}} = \{x \mid \exists y.(x,y) \in [\![R]\!]_{\mathsf{PExpr}} \cap [\![S]\!]_{\mathsf{PExpr}}\}$$

Regular XPath$^{\approx}$ can be viewed as a mix between Core XPath 1.0 and *regular path expressions* [1]: it has the filter expressions of the former and the Kleene star of the latter. It is still mainly studied in the theoretical community [9, 24, 7].

The query evaluation problem for Regular XPath$^{\approx}$ is in PTIME (in fact, in quadratic time), and the query containment problem is EXPTIME-complete [25].

3. EXPRESSIVITY OF XPATH DIALECTS

We have defined four XPath fragments. How do they compare in terms of expressivity and succinctness? We will answer this question by mapping each XPath dialect to an equally expressive variant of first-order logic.

Since the data model of an XML document is a finite sibling ordered tree, it is natural to consider first-order logic in the signature with eight atomic binary relations corresponding to the basic axes (`child`, `parent`, `left` and `right`, and their transitive closures `descendant`, `ancestor`, `following-sibling` and `preceding-sibling`) plus a unary predicate for each tag name. We will call the first order language in this signature FO_{tree}. With $FO_{\mathsf{tree}}(x)$ and $FO_{\mathsf{tree}}(x, y)$ we denote the FO_{tree} formulas in one and two free variables, respectively.

Besides looking at expressive power, we will also compare different languages in terms of *succinctness*. As usual, if two languages, L and L', are equally expressive, we say that L is *(at least) exponentially more succinct than* L' if there is a infinite sequence of L-expressions $R_1, R_2, \ldots$ where the length of R_k is polynomial in k, such that for every sequence of equivalent L'-expressions $R'_1, R'_2, \ldots$, the length of R'_k is exponential in k. Similarly, one can say that a language is *non-elementarily more succinct* than another language.

The results from this section are summarized in Table 3. These results hold both for path expressions and for node expressions.

The results discussed in this section naturally build on a existing line of research in temporal logic, which originates in the work of H. Kamp [15] and which studies expressive completeness for various temporal logics on trees. A survey of this area may be found in [13].

Core XPath 1.0

As we have already seen in Section 1, not every FO_{tree}-definable binary relation is definable in Core XPath 1.0. However, we can define a natural fragment of FO_{tree} with respect to which Core XPath 1.0 is complete.

Let $\exists FO^{(\mathrm{mon}\neg)}_{\mathsf{tree}}$ be the fragment of FO_{tree} where negation can only be applied to subformulas with exactly one free variable, and universal quantification is disallowed altogether (thus, the connectives are conjunction, disjunction, and existential quantification, plus negation of formulas with at most one free variable). It can be seen from Table 2 that Core XPath 1.0 path expressions can be translated into this fragment of FO_{tree} (indeed, the only form of negation present in Core XPath 1.0 is negation in filter expressions, which corresponds to negation of a formula in one free variable). A converse translation is possible as well, although it involves an exponential blow-up (recall the example we gave in the introduction):

Theorem 1 (Core XPath 1.0 $\equiv \exists FO^{(\mathbf{mon}\neg)}_{\mathsf{tree}}(x, y)$)

1. *There is a linear translation from Core XPath 1.0 path expressions to $\exists FO^{(mon\neg)}_{\mathsf{tree}}(x, y)$ formulas, and an exponential translation backwards.*
2. *Indeed, $\exists FO^{(mon\neg)}_{\mathsf{tree}}(x, y)$ formulas are exponentially more succinct than Core XPath 1.0 path expressions.*

PROOF. The difficult direction of (1) can be proved by induction on the nesting depth of negation, using the fact that positive existential fist-order formulas can be translated to Core XPath 1.0 path expressions at the cost of an exponential blowup [2, 11]. For the exponential difference in succinctness, see [25, Thm. 26]. □

An alternative characterization of Core XPath 1.0, in terms of conjunctive queries and the two-variable fragment of FO_{tree}, is given in [19].

Core XPath 2.0

In the case of Core XPath 2.0, there is a precise match with FO_{tree}, in terms of expressive power. In fact, this Codd-completeness has been one of the design considerations for XPath 2.0 [16]. Moreover, it turns out to hold already for the variable free fragment. Still, the presence of variables matters for the succinctness of the language.

For simplicity, we consider only path expressions that have no *free variables*. For a discussion of expressive completeness in the presence of free variables, see [8].

Theorem 2 (Core XPath 2.0 $\equiv FO_{\mathsf{tree}}(x, y)$)

1. *There are linear translations between Core XPath 2.0 path expressions and $FO_{\mathsf{tree}}(x, y)$ formulas.*
2. *There is a linear translation from variable free Core XPath 2.0 path expressions to $FO_{\mathsf{tree}}(x, y)$ formulas and a non-elementary translation backwards.*
3. *$FO_{\mathsf{tree}}(x, y)$ formulas are at least exponentially more succinct than variable free Core XPath 2.0 path expressions.*

PROOF. The linear translations are straightforward. A non-elementary translation from FO_{tree} to variable free Core XPath 2.0 is given in [18]. The exponential difference in succinctness between FO_{tree} and variable free Core XPath 2.0 holds already on linear orders (i.e., documents in which each node has at most one child) [12]. □

In fact, it was shown in [18] that a more modest extension of Core XPath 1.0 called *Conditional XPath* is already expressively complete for FO_{tree}. It extends Core XPath 1.0 with "conditional axes" of the form (Axis `while` NodeExpr), with Axis $\in \{\mathtt{child}, \mathtt{parent}, \mathtt{left}, \mathtt{right}\}$. Without going into further details, we only mention that (Axis `while` T) :: N can be written in Core XPath 2.0 as

$$\mathsf{Axis}^+ :: N \ \mathtt{except}\ (\mathsf{Axis}^+ :: *[\mathtt{not}(T)]/\mathsf{Axis}^+ :: *)$$

where Axis^+ is the transitive version of Axis.

Table 3: Expressivity and succinctness of XPath dialects.

XPath dialect	Core XPath 1.0	$\subsetneq$	Variable-free Core XPath 2.0	$\equiv$	Core XPath 2.0	$\subsetneq$	Regular XPath$^{\approx}$
Equivalent FO-dialect	$\exists FO^{\text{mon}\neg}_{\mathsf{tree}}$ (exponential succinctness gap)		FO_{tree} (at least exponential succinctness gap)		FO_{tree} (no succinctness gap: linear translations)		FO^{*}_{tree} (non-elementary succinctness gap)

Regular XPath$^{\approx}$

Since the conditional axes of [18] are definable in Regular XPath$^{\approx}$ using the Kleene star — `(Axis while` T`)::`N is equivalent to `(Axis::`$*$ `[not(`T`)])*/Axis::`N — we already know by [18] that Regular XPath$^{\approx}$ extends FO_{tree} in expressive power. In order to give a precise characterization of the expressive power of Regular XPath$^{\approx}$, we must consider an extension of FO_{tree}.

The simplest option is to simply extend FO_{tree} with a Kleene star (i.e., a transitive closure operator for binary relations). Thus, let $FO^{*}_{\mathsf{tree}}(x, y)$ be the extension of FO_{tree} (x, y) with a transitive closure operator that applies to formulas with exactly two free variables. Then the following is proved in [24] and [25, Thm. 27]:

Theorem 3 (Regular XPath$^{\approx}$ $\equiv FO^{*}_{\mathsf{tree}}$)

1. *There is a linear translation from Regular XPath$^{\approx}$ path expressions to $FO^{*}_{\mathsf{tree}}(x, y)$ formulas, and a non-elementary translation backwards.*
2. *In fact, $FO^{*}_{\mathsf{tree}}(x, y)$ formulas are non-elementarily more succinct than Regular XPath$^{\approx}$ path expressions.*

Incidentally, FO^{*}_{tree} is not the same as $FO_{\mathsf{tree}} + TC^{1}$: the standard unary transitive closure operator TC^{1} can be applied to formulas containing more than two free variables, as long as two of the variables are designated; the others are treated as parameters (cf. for instance [5]). We do not know at present whether FO^{*}_{tree} and $FO_{\mathsf{tree}} + TC^{1}$ have the same expressive power on trees.

4. ALGEBRAS FOR XPATH DIALECTS

The previous section showed that Core XPath 2.0 corresponds in expressive power to exactly first-order logic. The next question is which algebras are appropriate for representing query plans for Core XPath 2.0 expressions. The same question holds for the other dialects we discussed. Codd's relational algebra seems a natural choice because it is again equally expressive as first-order logic. Indeed, we will see that it is a good choice when considering Core XPath 2.0. For other XPath dialects however (including the variable free fragment of Core XPath 2.0), there are better options.

In Section 1.3 we gave criteria for determining whether an algebra is suitable for an XPath dialect. In this section, we discuss four different algebras, and determine which ones match best with each XPath dialect. The results are summarized in Table 5.

To simplify the presentation, we will first consider Core XPath 1.0 and 2.0, and only afterward Regular XPath$^{\approx}$, as the latter requires (a mild form of) recursion in the algebra.

4.1 Four candidate algebras

Codd's relational algebra (CRA) and its fragment CRA($mon\neg$)

We briefly recall Codd's relational algebra. A characteristic feature of this algebra is that it is *many sorted*: each expression has an associated *arity* corresponding to the number of columns of the table it computes. The atomic expressions are simply the names of the relations in the database, and the operations are *selection* (σ), *projection* (π), *cross-product* ($\times$), *union* ($\cup$) and *complementation* ($-$).

The fact that there is no bound on the arity of the expressions has some negative consequences on the complexity of query evaluation: it is PSPACE-complete, whereas it becomes polynomial if there is a bound on the allowed arity of (sub)expressions [3, 28].

Inspired by the results in the previous section, it makes sense to distinguish another restricted fragment of CRA, namely $CRA(mon\neg)$. This fragment is obtained by restricting the use of complementation to unary tables. Note that all $SPCU$-expressions still belong to this fragment.

Tarski's relation algebra (TRA)

Tarski's relation algebra [22, 23] is an algebra of binary relations: each expression denotes a table with precisely two columns. The operations on binary relations considered by Tarski are the Boolean operations (union, intersection and complementation), as well as composition $\circ$ and converse $(\cdot)^{-1}$. There are also two constants (or, 0-ary operations) $\top$ and ϵ, which stand for the total relation and the identity relation (over the given domain). A typical example of an equivalence in this algebra is $\alpha \circ (\beta \cup \gamma) \equiv \alpha \circ \beta \cup \alpha \circ \gamma$.

It was shown in [23] that TRA has the same expressive power as the three-variable fragment of first-order logic in two free variables, over vocabularies consisting of binary relations only.

Although in TRA all expressions denote binary relations, unary relations can be easily dealt with as

well, for instance by treating them as subrelations of the identity relation (e.g., $\{a, b, c\}$ can be treated as $\{(a, a), (b, b), (c, c)\}$).

Dynamic relation algebra (DRA)

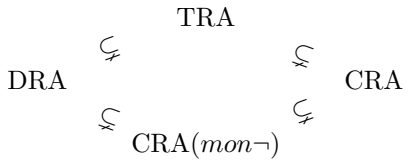
In [27, 14], a reduct of Tarski's relation algebra is studied containing only the operations $\cup$, $\circ$ and $\sim$. The latter of these is called the *counterdomain* operation. It takes a binary relation R and produces a subrelation of the identity relation: $\sim R$ denotes $\{(x, y) \mid x = y \text{ and } \neg\exists z.(x, z) \in R\}$. In TRA, it can be expressed as $\epsilon - (R \circ \top)$. This operator is quite handy: e.g., $\sim$`child` expresses "I am a leaf node", and $\sim\sim$`child` expresses "I am *not* a leaf node". We call this algebra *dynamic relation algebra* (DRA).

The signature of DRA might seem poor, but it is rich enough to capture all of Core XPath 1.0. If we encode properties of nodes as subrelations of the identity relation (as we already suggested above), then we have the following translation:

$$
\begin{array}{lcl}
\mathsf{TR}_{\mathsf{PExpr}}(\mathsf{Axis} :: *) & = & \mathsf{Axis} \\
\mathsf{TR}_{\mathsf{PExpr}}(\mathsf{Axis} :: N) & = & \mathsf{Axis} \circ N \\
\mathsf{TR}_{\mathsf{PExpr}}(R/S) & = & \mathsf{TR}_{\mathsf{PExpr}}(R) \circ \mathsf{TR}_{\mathsf{PExpr}}(S) \\
\mathsf{TR}_{\mathsf{PExpr}}(R \texttt{ union } S) & = & \mathsf{TR}_{\mathsf{PExpr}}(R) \cup \mathsf{TR}_{\mathsf{PExpr}}(S) \\
\mathsf{TR}_{\mathsf{PExpr}}(R[T]) & = & \mathsf{TR}_{\mathsf{PExpr}}(R) \circ \mathsf{TR}_{\mathsf{NExpr}}(T) \\
\mathsf{TR}_{\mathsf{NExpr}}(\mathsf{PathExpr}) & = & \sim\sim \mathsf{TR}_{\mathsf{PExpr}}(\mathsf{PathExpr}) \\
\mathsf{TR}_{\mathsf{NExpr}}(\texttt{not } T) & = & \sim \mathsf{TR}_{\mathsf{NExpr}}(T) \\
\mathsf{TR}_{\mathsf{NExpr}}(T_1 \texttt{ and } T_2) & = & \mathsf{TR}_{\mathsf{NExpr}}(T_1) \circ \mathsf{TR}_{\mathsf{NExpr}}(T_2) \\
\mathsf{TR}_{\mathsf{NExpr}}(T_1 \texttt{ or } T_2) & = & \mathsf{TR}_{\mathsf{NExpr}}(T_1) \cup \mathsf{TR}_{\mathsf{NExpr}}(T_2)
\end{array}
$$

In [27], an elegant model theoretic characterization of DRA is given in terms of *safety for bisimulations*.

DRA is a fragment of both TRA and CRA($mon\neg$). More precisely, the relationships between the four algebras on arbitrary models are as follows:



When we restrict attention to XML documents (i.e., where the atomic relations are the 8 binary relations corresponding to the different axes, as well a "unary" relation for each of the different tag names), the situation is a bit different: on this restricted class of models TRA and CRA have the same expressive power, as do DRA and CRA($mon\neg$).

4.2 Complexity of these algebras on trees

We will now discuss the complexity of *query evaluation* and *query containment* for the four algebras interpreted on XML-trees (i.e., where the atomic relations are the 8 binary relations corresponding to the different axes, as well a "unary" relation for each of the different tag names). As before, in the case of query evaluation we consider the combined complexity of testing whether a given pair belongs to the relation defined by a given path expression on a given XML document. Table 4 provides a summary of the results.

Table 4: Complexity of evaluation and containment for the algebras on trees.

	Evaluation	*Containment*
CRA	PSPACE-compl.	Non-elementary
$CRA(mon\neg)$	NP-hard, in P^{NP}	2-EXPTIME-compl.
TRA	PTIME (quadratic)	Non-elementary
DRA	PTIME (linear)	EXPTIME-compl.

Containment.

By Rabin's theorem, query containment is decidable for all four algebras. For TRA and CRA, containment is non-elementary, as follows from Stockmeyer's non-elementary lower bound for the non-emptiness problem of star-free expressions [21, 25]. The results for CRA($mon\neg$) and DRA follow from known results about XPath. In particular, the 2-EXPTIME-hardness of CRA($mon\neg$) query containment follows from the same lower bound for Core XPath 1.0 extended with path intersection, as the latter can be linearly translated into CRA($mon\neg$). The upper bound follows from the existence of a singly exponential translation from CRA($mon\neg$)-expressions of arity 2 to Core XPath 1.0, and the fact that Core XPath 1.0 has an EXPTIME-complete query containment problem (the restriction to expressions of arity 2 is not essential: containment of CRA($mon\neg$)-expressions of arity greater than 2 can be linearly reduced to containment of ones of arity 2, in fact to Boolean CRA($mon\neg$)-expressions) [25]. The result for DRA follows from linear translations to Core XPath 1.0.

Evaluation.

The combined complexity of query evaluation for CRA is PSPACE-complete, also when restricted to XML-trees [3]. As TRA corresponds to a fixed variable fragment of first-order logic the complexity drops to PTIME. Using the bottom-up algorithm sketched in [28] it can be shown to be in $O(n^2)$. In [10], it is shown that query evaluation for Core XPath 1.0 can be performed in linear time. Because Core XPath 1.0 and DRA linearly translate to each other, the result transfers to DRA. Recall that we are not talking about the complexity of computing the relation denoted by a path expression (which could be quadratic in the size of the tree), but of the complexity of checking whether a given pair of nodes belongs to the denotation of a given expression in a given tree. Query evaluation for CRA($mon\neg$) is NP-hard: this holds even for positive conjunctive queries with only downward axis relations [11]. For the P^{NP}-upperbound, we use an algorithm that runs in polynomial time and that uses an oracle for testing whether a tuple belongs to the answer set of an SPCU-expression. The algorithm proceeds roughly as follows: given an expression α, it starts by listing all subexpressions whose main connective is a (unary) complementation operator, in order of growing length.

Table 5: Which algebra for which XPath dialect?

	CRA	CRA($mon\neg$)	TRA	DRA
Core XPath 1.0	**Y** (linear translation) **N** (too expressive) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**Y** (linear translation) **N** (too expressive) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **Y** (same complexity)
Core XPath 2.0 w/o variables	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**N** (no translation possible) **N** (too little expressivity) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **Y** (same complexity)	**N** (no translation possible) **N** (too little expressivity) **Y** (lower complexity)
Core XPath 2.0 with variables	**Y** (linear translation) **Y** (same expressivity) **Y** (same complexity)	**N** (no translation possible) **N** (too little expressivity) **Y** (lower complexity)	**N** (no elem. translation) **Y** (same expressivity) **Y** (same complexity)	**N** (no translation possible) **N** (too little expressivity) **Y** (lower complexity)
	CRA($*$)	CRA($mon\neg, *$)	TRA($*$)	DRA($*$, loop)
Regular XPath$^{\approx}$	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **Y** (same complexity)

One by one, it computes for each such subexpression α the (polynomially large) answer set, by asking the oracle for each element whether it belongs to the answer set. The occurrences of α within larger expressions are then replaced by the computed answer set. Finally, we are left with a single SPCU-expression, to which the oracle is once more applied.

4.3 Axiomatizations

One of our criteria for being a good algebra was the availability of an axiomatization of the valid equations on XML-trees (finite sibling ordered node-labeled trees). Only a few results are known here. In [2], an axiomatization is given for the $\sim$-free reduct of Dynamic Relation Algebras DRA with only the two downward axis plus atomic label tests. An axiomatization of the full DRA on XML-trees is not known (a complete axiomatization on arbitrary models is given in [14]). In [26], an axiomatization of first-order logic on XML-trees is given, from which an axiomatization for TRA on XML-trees is derived. We believe that in a similar way an axiomatization of CRA on XML-trees can be found. The TRA axiomatization consists of general axioms for the TRA similarity type like $R \circ (S \circ T) = (R \circ S) \circ T$ plus special axioms which are only valid on trees. Two examples are Tr5 and Tr11:

$$\text{Tr5.} \quad \downarrow^+ \circ \uparrow^+ \quad \equiv \quad \downarrow^+ [\downarrow] \cup \epsilon[\downarrow] \cup (\epsilon[\downarrow] \circ \uparrow^+)$$

$$\text{Tr11.} \quad \epsilon \cup \uparrow^+ \cup \downarrow^+ \cup (\uparrow^* \circ \rightarrow^+ \circ \downarrow^*) \cup (\uparrow^* \circ \leftarrow^+ \circ \downarrow^*) \quad \equiv \quad \top$$

Here we abbreviate the steps in the trees by arrows, e.g., $\downarrow$ is the `child` axis, $\uparrow$ is `parent`, etc. E.g., in XPath notation, the left-hand side of Tr5 would be `descendant/ancestor`. Also, we use $R[S]$ as a shorthand for $R \circ \sim\!\sim S$. Tr5 is a natural complexity reducing equivalence when read from left to right. Tr11 states the well known fact that the self, ancestor, descendant, following and preceding axis relations partition each XML-tree from every given node.

4.4 Which algebra for which XPath?

We now have three XPath dialects (Regular XPath$^{\approx}$ will be dealt with in the next subsection) and four candidate algebras. We determine which algebra fits best to which fragment by answering the following questions, corresponding to the first three requirements from Section 1.3:

1. Is there a linear translation from the expressions in XPath dialect to expressions in the algebra?
2. Are the XPath dialect and the algebra equally expressive?
3. Do the XPath dialect and the algebra have the same query containment and evaluation complexities?

(as for the fourth requirement, concerning the existence of nice sets of algebraic equivalence rules for the algebra, we have too little information at present to say much about it).

The answers, based on the results discussed in the previous sections, are given in Table 5. The combinations with only affirmative answers are marked by a gray background.

4.5 Regular XPath$^{\approx}$

For Regular XPath$^{\approx}$, the algebras need to be extended with a transitive closure operator. In the case of TRA and DRA, the semantics of such an operator is clear: the denotation of R^* is the reflexive, transitive closure of the binary relation denoted by R. In the case of CRA and CRA($mon\neg$) a similar proviso needs to be made as for FO^*_{tree} (cf. Section 3): the transitive closure operator may only be applied to expressions that denote tables with precisely two columns. We use

CRA(∗), CRA($mon\neg, *$), TRA(∗) and DRA(∗) to denote the extensions of the respective algebras with the transitive closure operator, conform this restriction.

The path equalities of Regular XPath$^{\approx}$ can be expressed in CRA(∗) and CRA($mon\neg, *$) using intersection and projection, and in TRA(∗) using intersection and $\sim$: $R \approx S$ can be expressed as $\sim\sim(R \cap S)$. On the other hand, in DRA(∗) it is not clear whether path equalities can be expressed. Let DRA(∗,loop) denote the extension of DRA with both the Kleene star and the $(\cdot)^{\mathsf{loop}}$ operator, that has the following semantics [9]: $R^{\mathsf{loop}} = R \cap \epsilon$. Using loop, and given the fact that Regular XPath$^{\approx}$ is closed under taking inverses of path expressions, we can express path equalities: $R \approx S$ translates to $(R \circ S^{-1})^{\mathsf{loop}}$.

It follows from Theorem 3 that Regular XPath$^{\approx}$, DRA(∗,loop), TRA(∗), CRA(∗) and CRA($mon\neg, *$) all have the same expressive power. Of these four, DRA(*,loop) is the most suitable algebra for Regular XPath$^{\approx}$, since its query containment problem is ExpTime-complete (as follows from the fact that there are linear translations from and to Regular XPath$^{\approx}$ [25]). See also Table 5.

5. CONCLUSION AND OPEN PROBLEMS

We have discussed four dialects of Navigational XPath, and we have shown that they correspond, in terms of expressive power, to natural fragments or extensions of first-order logic. Furthermore, we have identified suitable algebras for each of the dialects.

We have not discussed *monadic second-order logic* (MSO) as a target for expressive power. Several dialects of navigational XPath have been proposed in the literature that have the same expressive power as MSO (see for instance [9, 24]), but in our view none has the simplicity and appeal of the dialects we studied here.

We end with four open problems. The first two are related to the DRA and its extension DRA(∗,loop). The second two relate to the strictness of the hierarchy of path languages given in [5].

Problem 1 *Give axiomatizations for DRA and DRA(∗,loop) on XML-trees.*

Problem 2 *Does loop really any add expressive power to DRA(∗,loop)? Or equivalently, do path equalities really contribute to the expressive power of Regular XPath$^{\approx}$?*

Problem 3 *Is Regular XPath$^{\approx}$ (or equivalently FO^{*}_{tree}) less expressive than MSO?*

Problem 4 *Is FO^{*}_{tree} less expressive than $FO_{\mathsf{tree}} + TC^{1}$? If so, identify an XPath dialect that has exactly the same expressive power as the latter.*

Acknowledgments. We are grateful to Loredana Afanasiev and Tadeusz Litak for helpful comments. Balder ten Cate is supported by NWO research grant 639.021.508.

6. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the web.* Morgan Kaufman, 2000.

[2] M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. In *Proc. ICDT 2003*, 2003.

[3] A. Chandra and D. Harel. Structure and complexity of relational queries. *J. Comput. Syst. Sci.*, 25(1):99–128, 1982.

[4] E. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Database Systems*, pages 33–64. Prentice-Hall, 1972.

[5] J. Engelfriet and H. Hoogeboom. Nested pebbles and transitive closure. In *Proc. STACS*, pages 477–488, 2006.

[6] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. SMOQE: a system for providing secure access to XML. In *Proc. VLDB'2006*, pages 1227–1230, 2006.

[7] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Rewriting regular XPath queries on XML views. In *Proc. ICDE'2007*, 2007.

[8] E. Filiot, J. Niehren, J.-M. Talbot, and S. Tison. Polynomial time fragments of xpath with variables. In *Proc. PODS'07*, 2007.

[9] E. Goris and M. Marx. Looping caterpillars. In *Proc. LICS 2005*. IEEE Computer Society, 2005.

[10] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *VLDB'02*, 2002.

[11] G. Gottlob, C. Koch, and K. Schulz. Conjunctive queries over trees. In *Proc. PODS'04*, pages 189–200, 2004.

[12] M. Grohe and N. Schweikardt. The succinctness of first-order logic on linear orders. 1(1), 2005.

[13] I. Hodkinson and M. Reynolds. Separation - past, present, and future. In S. Artemov et al., editor, *We will show them! (Essays in honour of Dov Gabbay on his 60th birthday)*, pages 117–142. College Publications, 2005.

[14] M. Hollenberg. An equational axiomatization of dynamic negation and relational composition. *Journal of Logic, Language and Information*, 6(4):381–401, 1997.

[15] J.A.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[16] M. Kay. *XPath 2.0 Programmer's Reference*. Wrox, 2004.

[17] M. Marx. XPath with conditional axis relations. In *Proc. EDBT'04*, volume 2992 of *LNCS*, pages 477–494, 2004.

[18] M. Marx. Conditional XPath. *ACM Transactions on Database Systems (TODS)*, 30(4):929–959, 2005.

[19] M. Marx and M. de Rijke. Semantic Characterizations of Navigational XPath. *SIGMOD Record*, 34(2):41–46, 2005.

[20] F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *Proc. ICDT 2003*, 2003.

[21] L. Stockmeyer. *The Complexity of Decision Problems in Automata Theory*. PhD thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., 1974.

[22] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6:73–89, 1941.

[23] A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*, volume 41. AMS Colloquium publications, Providence, Rhode Island, 1987.

[24] B. ten Cate. The expressivity of XPath with transitive closure. In *Proc. PODS*, pages 328–337, 2006.

[25] B. ten Cate and C. Lutz. The complexity of query containment in expressive fragments of XPath 2.0. In *Proc. PODS'07*, 2007.

[26] B. ten Cate and M. Marx. Axiomatizing the logical core of XPath 2.0. In *Proc. ICDT'07*, 2007.

[27] J. van Benthem. Program constructions that are safe for bisimulation. *Studia Logica*, 60(2):331–330, 1998.

[28] M. Vardi. On the complexity of bounded–variable queries. In *Proc. PODS'95*, pages 266–276, 1995.