# XPath with conditional axis relations

Maarten Marx[⋆] marx@science.uva.nl

Language and Inference Technology, ILLC, Universiteit van Amsterdam, The Netherlands.

**Abstract.** This paper is about the W3C standard node-addressing language for XML documents, called XPath. XPath is still under development. Version 2.0 appeared in 2001 while the theoretical foundations of Version 1.0 (dating from 1998) are still being widely studied. The paper aims at bringing XPath to a "stable fixed point" in its development: a version which is expressively complete, still manageable computationally, with a user-friendly syntax and a natural semantics. We focus on an important axis relation which is not expressible in XPath 1.0 and is very useful in practice: *the conditional axis*. With it we can express paths specified by for instance "do a `child` step, while `test` is true at the resulting node". We study the effect of adding conditional axis relations to XPath on its expressive power and the complexity of the query evaluation and query equivalence problems. We define an XPath dialect $\mathcal{X}$CPath which is expressively complete, has a linear time query evaluation algorithm and for which query equivalence given a DTD can be decided in exponential time.

## 1 Introduction

XPath 1.0 [38] is a variable free language used for selecting nodes from XML documents. XPath plays a crucial role in other XML technologies such as XSLT [42], XQuery [41] and XML schema constraints, e.g., [40]. The latest version of XPath (version 2.0) [39] is much more expressive and is close to being a full fledged tree query language. Version 2.0 contains variables which are used in if–then–else, for and quantified expressions. The available axes are the same in both versions.

The purpose of this paper is to show that useful and more expressive variants of XPath 1.0 can be created without introducing variables. We think that the lack of variables in XPath 1.0 is one of the key features for its success, so we should not lightly give it up.

This paper uses the abstraction to the logical core of XPath 1.0 developed in [17,16]. This means that we are discussing the expressive power of the language on XML document tree models. The central expression in XPath is $\mathsf{axis} :: node\_label[\mathsf{filter}]$ (called a *location path*) which when evaluated at node $n$ yields an answer set consisting of nodes $n'$ such that

– the axis relation goes from $n$ to $n'$,
– the node tag of $n'$ is $node\_label$, and
– the expression filter evaluates to true at $n'$.

[⋆] Research supported by NWO grant 612.000.106. Thanks to Patrick Blackburn, Linh Nguyen and Petrucio Viana.

We study the consequences of varying the set of available axis relations on the expressive power and the computational complexity of the resulting XPath dialect. All axis relations of XPath 1.0, like child, descendant etc., are assumed as given. We single out an important axis relation which is not expressible in XPath 1.0 and is very useful in practice: *the conditional axis*. With it we can express paths specified by for instance "do a `child` step, while `test` is true at the resulting node". Such conditional axes are widely used in programming languages and temporal logic specification and verification languages. In temporal logic they are expressed by the Since and Until operators. The acceptance of temporal logic as a key verification language is based on two interrelated results. First, as shown by Kamp [21] and later generalized and given an accessible proof by Gabbay, Pnueli, Shelah and Stavi [14], temporal logic over linear structures can express every first order definable property. This means that the language is in a sense complete and finished. This is an important point in the development of a language. It means that no further expressivity (up to first order of course) needs to be added. The second important result concerned the complexity of the model checking and validity problem. Both are complete for polynomial space, while model checking can be done in time $O(2^{|Q|} \cdot |D|)$, with $|Q|$ and $|D|$, the size of the query and data, respectively [8].

The paper aims at bringing navigational XPath to a similar " fixed point" in its development: a version which is expressively complete, still manageable computationally, with a user-friendly syntax and a natural semantics. The key message of the paper is that all of this is obtainable by adding the conditional axes to the XPath syntax. Our main contributions are the following:

– A motivation of conditional paths with natural examples (Section 2).
– The definition of several very expressive variable free XPath dialects, in particular the language $\mathcal{X}$CPath (Section 3).
– A comparison of the expressive power of the different dialects (Section 4).
– An investigation of the computational complexity of the discussed dialects. The main results are that the extra expressivity comes at no (theoretical) extra cost: query evaluation is in linear time in the size of the query and the data (Section 5), and query equivalence given a DTD is decidable in exponential time (Section 6).

For succinctness and readability all proofs are put in the Appendix.

We end this introduction with a few words on related literature. The observation that languages for XML like XPath, DTD's and XSchema can be viewed as propositional temporal, modal or description logics has been made by several authors. For instance, Miklau and Suciu [25] and Gottlob et al [15] embed XPath into CTL. The group round Calvanese, de Giacomo and Lenzerini published a number of papers relating DTD's and XPath to description logic, thereby obtaining powerful complexity results cf, e.g., [7]. Demri, de Rijke and Alechina reduce certain XML constraint inference problems to propositional dynamic logic [2]. The containment problem for XPath given a set of constraints (a DTD for instance) has been studied in [36,9,25,26]. The complexity of XPath query evaluation was determined in [15,16]. We are convinced that this list is incomplete, if only for the sole reason that the connection between the formalisms and the structures in which they are interpreted is so obvious. Our work differs from most in that we consider node labeled trees, whereas (following [1]) most authors model semistructured data as edge labeled trees. This is more than just a different semantic viewpoint. It allows us to use the same syntax as standard XPath and to give simpler and more perspicuous embeddings into known formalisms. Taking this viewpoint it turns out that many results on XPath follow easily from known results: linear time query evaluation follows directly from linear time model checking for propositional dynamic logic (PDL); the exponential time query equivalence algorithm can also be deduced (with quite a bit more work) from results on PDL, and finally Gabbay's separation technique for temporal logic is directly applicable to node labeled sibling ordered trees.

## 2 Conditional paths

This section motivates the addition of conditional paths to the XPath language.

**Example 1.** Consider an XML document containing medical data as in Figure 1. Each node describes a person with an attribute assigning a name and an attribute stating whether or not the person has or had leukemia. The child-of relation in the tree models the real child-of relation between persons. Thus the root of the tree is person `a` which has leukemia. Person `a` has a child `a1` without the disease and a grandchild `a12` which has it.

```
<P name=a leukemia=yes>
   <P name=a1 leukemia=no>
       <P name=a11 leukemia=no/>
       <P name=a12 leukemia=yes/>
       <P name=a13 leukemia=no/>
   </P>
   <P name=a2 leukemia=yes>
       <P name=a21 leukemia=yes/>
       <P name=a22 leukemia=no/>
   </P>
</P>
```

**Fig. 1.** XML document containing medical data.

Consider the following information need: *given a person $x$, find descendants $y$ of $x$ without leukemia such that all descendants of $x$ between $x$ and $y$ had/have leukemia*. The answer set of this query in the example document when evaluated at the root is the set of nodes with name attribute `a1` and `a22`. When evaluated at node `a1`, the answer set is $\{$`a11,a13`$\}$, when evaluated at `a2` the answer set is $\{$`a22`$\}$, and at all other nodes, the answer set is empty. The information need can be expressed in first order logic using a suitable signature. Let `child` and `descendant` be binary and `P` and `has_leukemia` be unary predicates. The information need is expressed by a first order formula in two free variables:

$$\texttt{descendant}(x,y) \wedge \neg\texttt{has\_leukemia}(y) \wedge$$
$$\forall z((\texttt{descendant}(x,z) \wedge \texttt{descendant}(z,y)) \rightarrow \texttt{has\_leukemia}(z)).$$

This information need can be expressed in XPath 1.0 for arbitrarily deep documents by the infinite disjunction (for readability, we abbreviate *leukemia* to *l)*

```
child::P[@l='no'] |
child::P[@l='yes']/child::P[@l='no'] |
child::P[@l='yes']/child::P[@l='yes']/child::P[@l='no'] ...
```

Theorem 3 below states that it is not possible to express this information need in Core XPath. What seems to be needed is the notion of a *conditional path*. Let $_{[\texttt{test}]}\texttt{child}$ denote all pairs $(n, n')$ such that $n'$ is a child of $n$ and the `test` succeeds at $n$. Then the information need can be expressed by

$$\texttt{child} :: \texttt{P}/({}_{[@\texttt{l}='\texttt{yes}']}\texttt{child})^* :: \texttt{P}[@\texttt{l} =' \texttt{no}'].$$

The axis $({}_{[@\texttt{l}='\texttt{yes}']}\texttt{child})^*$ describes the reflexive transitive closure of $_{[@\texttt{l}='\texttt{yes}']}\texttt{child}$, whence either the path self or a child–path $n_0, n_1, \ldots, n_k$, for $k \geq 1$ such that at all nodes $n_0, \ldots n_{k-1}$ the leukemia attribute equals `yes`.

The next example discusses an information need which is not expressible using conditional paths. Admittedly the example is rather contrived. In fact it is very difficult to find natural information needs on the data in Figure 1 which are not expressible using transitive closure over conditional paths. Theorem 4 below states that every first order expressible set of nodes can be described as an XPath expression with transitive closure over conditional paths. Thus the difficulty in finding not expressible natural queries arises because such queries are genuinely second order. We come back to this point in Section 4 when we discuss the expressive power of XPath dialects.

**Example 2.** Consider the query "find all descendants which are an even number of steps away from the node of evaluation", expressible as $(\texttt{child}; \texttt{child})^* :: *$. Note that the axis relation contains a transitive closure over a sequence of atomic paths. The proof of Theorem 3 shows that this information need is not expressible using just transitive closure over conditional paths. The information need really expresses a second order property.

## 3 A brief introduction to XPath

[16] proposes a fragment of XPath 1.0 which can be seen as its logical core, but lacks many of the functionality that account for little expressive power. In effect it supports all XPath's axis relations, except the attribute relation[1], it allows sequencing and taking unions of path expressions and full booleans in the filter expressions. It is called Core XPath. A similar logical abstraction is made in [4]. As the focus of this paper is expressive power, we also restrict XPath to its logical core.

We will define four XPath languages which only differ in the axis relations allowed in their expressions. As in XPath 1.0, we distinguish a number of axis relations. Instead of the rather verbose notation of XPath 1.0, we use a self-explanatory graphical notation, together with regular expression operators $^+$ and $^*$. For the definition of the XPath languages, we follow the presentation of XPath in [16]. The expressions obey

[1] This is without loss of generality as instead of modeling attributes as distinct axes, as in the standard XML model, we may assign multiple labels to each node, representing whether a certain attribute-value pair is true at that node.

the standard W3C unabbreviated XPath 1.0 syntax, except for the different notation of the axis relations. The semantics is as in [4] and [15], which is in line with the standard XPath semantics from [34].

Our simplest language $\mathcal{X}\mathsf{Core}$ is slightly more expressive than Core XPath (cf. Remark 1). We view $\mathcal{X}\mathsf{Core}$ as the baseline in expressive power for XPath languages. $\mathcal{X}\mathsf{CPath}$, for *conditional* XPath, simply extends $\mathcal{X}\mathsf{Core}$ with conditional paths. The key difference between $\mathcal{X}\mathsf{Core}$ and the three other languages is the use of filter expressions inside the axis relations, in particular as conditional paths. Regular expressions[2] with tests are well studied and often applied in languages for specifying paths (see the literature on Propositional Dynamic Logic (PDL) and Kleene algebras with tests [18,23]).

**Definition 1** The syntax of the XPath languages $\mathcal{X}\mathsf{Core}, \mathcal{X}\mathsf{CPath}, \mathcal{X}_{reg}^{\mathsf{CPath}}, \mathcal{X}_{reg}$ is defined by the grammar

locpath ::= axis '::'ntst | axis '::' ntst '['fexpr']' | '/' locpath | locpath '/' locpath |
    locpath '|' locpath
fexpr ::= locpath | $\mathtt{not}$ fexpr | fexpr $\mathtt{and}$ fexpr | fexpr $\mathtt{or}$ fexpr
prim_axis ::= $\mathsf{self} \mid \Downarrow \mid \Uparrow \mid \Rightarrow \mid \Leftarrow$

| | |
|---|---|
| $\mathcal{X}\mathsf{Core}$ | $\text{axis} ::= \text{prim\_axis} \mid \text{axis}^*$ |
| $\mathcal{X}\mathsf{CPath}$ | $\text{axis} ::= \text{prim\_axis} \mid {}_{\text{fexpr}}\text{prim\_axis} \mid \text{prim\_axis}_{\text{fexpr}} \mid \text{axis}^*$ |
| $\mathcal{X}_{reg}^{\mathsf{CPath}}$ | $\text{axis} ::= \text{prim\_axis} \mid {}_{\text{fexpr}}\text{prim\_axis} \mid \text{prim\_axis}_{\text{fexpr}} \mid \text{axis};\text{axis} \mid \text{axis} \cup \text{axis} \mid \text{axis}^*$ |
| $\mathcal{X}_{reg}$ | $\text{axis} ::= \text{prim\_axis} \mid ?\text{fexpr} \mid \text{axis};\text{axis} \mid \text{axis} \cup \text{axis} \mid \text{axis}^*.$ |

where "locpath" (pronounced as *location path*) is the start production, "axis" denotes axis relations and "ntst" denotes tags labeling document nodes or the star '*' that matches all tags (these are called node tests). The "fexpr" will be called *filter expressions* after their use as filters in location paths. With an XPath expression we always mean a "locpath".

The semantics of XPath expressions is given with respect to an XML document modeled as a finite *node labeled sibling ordered tree*[3] (tree for short). Each node in the tree is labeled with a name tag from some alphabet. Sibling ordered trees come with two binary relations, the child relation, denoted by $R_{\Downarrow}$, and the immediate_right_sibling relation, denoted by $R_{\Rightarrow}$. Together with their inverses $R_{\Uparrow}$ and $R_{\Leftarrow}$ they are used to interpret the axis relations.

Each location path denotes a binary relation (a set of paths). The meaning of the filter expressions is given by the predicate $\mathcal{E}(n, \mathsf{fexpr})$ which assigns a boolean value. Thus a filter expression $\mathsf{fexpr}$ is most naturally viewed as denoting sets of nodes: all $n$ such that $\mathcal{E}(n, \mathsf{fexpr})$ is true. For examples, we refer to Section 2 and to [16]. Given a tree $\mathfrak{M}$ and an expression $f$, the denotation or meaning of $f$ in $\mathfrak{M}$ is written as $[\![ f ]\!]_{\mathfrak{M}}$. Table 1 contains the definition of $[\![ \cdot ]\!]_{\mathfrak{M}}$.

[2] Regular expressions are strings generated by the grammar $r ::= \epsilon \mid a \mid r^+ \mid r^* \mid r;r \mid r \cup r$ with $a$ a primitive symbol and $\epsilon$ the empty string. The operations have their usual meaning.

[3] A sibling ordered tree is a structure isomorphic to $(N, R_{\Downarrow}, R_{\Rightarrow})$ where $N$ is a 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 $s \cdot k - 1 \in N$. For $n, n' \in N$, $nR_{\Downarrow}n'$ holds iff $n' = n \cdot k$ for $k$ a natural number; $nR_{\Rightarrow}n'$ holds iff $n = s \cdot k$ and $n' = s \cdot k + 1$.

$$\begin{array}{lcl}
[\![\mathcal{X} :: t]\!]_{\mathfrak{M}} & = & \{(n,n') \mid n[\![\mathcal{X}]\!]_{\mathfrak{M}} n' \text{ and } t(n')\} \\
[\![\mathcal{X} :: t[e]]\!]_{\mathfrak{M}} & = & \{(n,n') \mid n[\![\mathcal{X}]\!]_{\mathfrak{M}} n' \text{ and } t(n') \text{ and } \mathcal{E}_{\mathfrak{M}}(n',e)\} \\
[\![/\text{locpath}]\!]_{\mathfrak{M}} & = & \{(n,n') \mid (root,n') \in [\![\text{locpath}]\!]_{\mathfrak{M}}\} \\
[\![\text{locpath}/\text{locpath}]\!]_{\mathfrak{M}} & = & [\![\text{locpath}]\!]_{\mathfrak{M}} \circ [\![\text{locpath}]\!]_{\mathfrak{M}} \\
[\![\text{locpath} \mid \text{locpath}]\!]_{\mathfrak{M}} & = & [\![\text{locpath}]\!]_{\mathfrak{M}} \cup [\![\text{locpath}]\!]_{\mathfrak{M}}
\end{array}$$

$$\begin{array}{lcl}
\mathcal{E}_{\mathfrak{M}}(n,\text{locpath}) = true & \iff & \exists n' : (n,n') \in [\![\text{locpath}]\!]_{\mathfrak{M}} \\
\mathcal{E}_{\mathfrak{M}}(n,\text{fexpr}_1 \texttt{ and } \text{fexpr}_2) = true & \iff & \mathcal{E}_{\mathfrak{M}}(n,\text{fexpr}_1) = true \text{ and } \mathcal{E}_{\mathfrak{M}}(n,\text{fexpr}_2) = true \\
\mathcal{E}_{\mathfrak{M}}(n,\text{fexpr}_1 \texttt{ or } \text{fexpr}_2) = true & \iff & \mathcal{E}_{\mathfrak{M}}(n,\text{fexpr}_1) = true \text{ or } \mathcal{E}_{\mathfrak{M}}(n,\text{fexpr}_2) = true \\
\mathcal{E}_{\mathfrak{M}}(n,\texttt{not } \text{fexpr}) = true & \iff & \mathcal{E}_{\mathfrak{M}}(n,\text{fexpr}) = false.
\end{array}$$

$$\begin{array}{lllll}
[\![\Downarrow]\!]_{\mathfrak{M}} & := R_{\Downarrow} & \quad & [\![\mathsf{self}]\!]_{\mathfrak{M}} & := \{(x,y) \mid x = y\} \\
[\![\Rightarrow]\!]_{\mathfrak{M}} & := R_{\Rightarrow} & & [\![p/q]\!]_{\mathfrak{M}} & := [\![p]\!]_{\mathfrak{M}} \circ [\![q]\!]_{\mathfrak{M}} \\
[\![\Uparrow]\!]_{\mathfrak{M}} & := R_{\Downarrow}^{-1} & & [\![p \mid q]\!]_{\mathfrak{M}} & := [\![p]\!]_{\mathfrak{M}} \cup [\![q]\!]_{\mathfrak{M}} \\
[\![\Leftarrow]\!]_{\mathfrak{M}} & := R_{\Rightarrow}^{-1} & & [\![p^*]\!]_{\mathfrak{M}} & := [\![\mathsf{self}]\!]_{\mathfrak{M}} \cup [\![p]\!]_{\mathfrak{M}} \cup [\![p]\!]_{\mathfrak{M}} \circ [\![p]\!]_{\mathfrak{M}} \cup \ldots \\
[\![p_{\text{fexpr}}]\!]_{\mathfrak{M}} & := [\![p]\!]_{\mathfrak{M}} \circ [\![?\text{fexpr}]\!]_{\mathfrak{M}} & & [\![?e]\!]_{\mathfrak{M}} & := \{(x,y) \mid x = y \text{ and } \mathcal{E}_{\mathfrak{M}}(x,e) = true\}. \\
[\![{}_{\text{fexpr}}p]\!]_{\mathfrak{M}} & := [\![?\text{fexpr}]\!]_{\mathfrak{M}} \circ [\![p]\!]_{\mathfrak{M}} & & &
\end{array}$$

**Table 1.** The semantics of $\mathcal{X}$Core and $\mathcal{X}$CPath.

*Remark 1.* XPath 1.0 (and hence Core XPath) has a strange asymmetry between the vertical (parent and child) and the horizontal (sibling) axis relations. For the vertical direction, both transitive and reflexive–transitive closure of the basic steps are primitives. For the horizontal direction, only the transitive closure of the immediate_left_ and immediate_right_sibling axis are primitives (with the rather ambiguous names *following* and *preceding_sibling*). $\mathcal{X}$Core removes this asymmetry and has all four one step navigational axes as primitives. XPath 1.0 also has two primitive axis related to the document order and transitive closures of one step navigational axes. These are just syntactic sugar, as witnessed by the following definitions:

$$\begin{array}{lcl}
\texttt{descendant} :: t[\phi] & \equiv & \Downarrow :: */\Downarrow^* :: t[\phi] \\
\texttt{following} :: t[\phi] & \equiv & \Uparrow^* :: */\Rightarrow :: */\Rightarrow\ast :: */\Downarrow^* :: t[\phi] \\
\texttt{preceding} :: t[\phi] & \equiv & \Uparrow^* :: */\Leftarrow :: */\Leftarrow^* :: */\Downarrow^* :: t[\phi].
\end{array}$$

So we can conclude that $\mathcal{X}$Core is at least as expressive as Core XPath, and has a more elegant set of primitives.

The reader might wonder why the set of XPath axes was not closed under taking converses. The next theorem states that this is not needed.

**Theorem 2.** *The set of axes of all four XPath languages are closed under taking converses.*

## 4 Expressive power

This section describes the relations between the four defined XPath dialects and two XPath dialects from the literature. We also embed the XPath dialects into first order and monadic second order logic of trees and give a precise characterization of the conditional path dialect $\mathcal{X}$CPath in terms of first order logic.

Core XPath [17] was introduced in the previous section. [4] considers the Core XPath fragment $\mathcal{X}^{\uparrow}_{r,[]}$ obtained by deleting the sibling relations and the booleans on the filter expressions.

**Theorem 3.** *1. $\mathcal{X}^{\uparrow}_{r,[]}$ is strictly contained in Core XPath, and Core XPath is strictly contained in $\mathcal{X}$Core.*
*2. $\mathcal{X}$Core is strictly contained in $\mathcal{X}$CPath.*
*3. $\mathcal{X}$CPath is strictly contained in $\mathcal{X}_{reg}$.*
*4. $\mathcal{X}_{reg}$ and $\mathcal{X}_{reg}^{\mathsf{CPath}}$ are equally expressive.*

We now view the XPath dialects as query languages over trees, and compare them to first and second order logic interpreted on trees. Before we can start, we must make clear what kind of queries XPath expresses.

In the literature on XPath it is often tacitly assumed that each expression is always evaluated at the root. Then the meaning of an expression is naturally viewed as a *set of nodes*. Stated differently, it is a query with one variable in the select clause. This tacit assumption can be made explicit by the notion of an *absolute* XPath expression /`locpath`. The answer set of /`locpath` evaluated on a tree $\mathfrak{M}$ (notation: $answer_{\mathfrak{M}}(/\texttt{locpath})$) is the set $\{n \in \mathfrak{M} \mid (root, n) \in [\![\texttt{locpath}]\!]_{\mathfrak{M}}\}$. The relation between filter expressions and arbitrary XPath expressions evaluated at the root becomes clear by the following equivalence. For each filter expression `fexpr`, $\mathcal{E}_{\mathfrak{M}}(n, \texttt{fexpr})$ is true if and only if $n \in answer_{\mathfrak{M}}(/\Downarrow^* :: *[\texttt{fexpr}])$. On the other hand, looking at Table 1 it is immediate that in general an expression denotes a *binary relation*.

Let $\mathcal{L}^{tree}_{FO}$ and $\mathcal{L}^{tree}_{MSO}$ be the first order and monadic second order languages in the signature with two binary relation symbols $Descendant$ and $Sibling$ and countably many unary predicates $P, Q, \ldots$ Both languages are interpreted on node labeled sibling ordered trees in the obvious manner: $Descendant$ is interpreted as the descendant relation $R^+_{\Downarrow}$, $Sibling$ as the strict total order $R^+_{\Rightarrow}$ on the siblings, and the unary predicates $P$ as the sets of nodes labeled with $P$. For the second order formulas, we always assume that all second order variables are quantified. So a formula in two free variables means a formula in two free variables ranging over nodes.

It is not hard to see that every $\mathcal{X}_{reg}$ expression is equivalent[4] to an $\mathcal{L}^{tree}_{MSO}$ formula in two free variables, and that every filter expression is equivalent to a formula in one free variable. $\mathcal{X}_{reg}$ can express truly second order properties as shown in Example 2. A little bit harder is

**Proposition 1.** *Every $\mathcal{X}$CPath (filter) expression is equivalent to an $\mathcal{L}^{tree}_{FO}$ formula in two (one) free variable(s).*

The converse of this proposition would state that $\mathcal{X}$CPath is powerful enough to express every first order expressible query. For one variable queries on Dedekind complete linear structures, the converse is known as Kamp's Theorem [21]. Kamp's result is generalized to other linear structures and given a simple proof in the seminal paper [14]. [24] showed that the result can further be generalized to sibling ordered trees:

[4] In fact, there is a straightforward logspace translation, see the proof of Theorem 7.(i).

**Theorem 4** **([24]).** *Every $\mathcal{L}_{FO}^{tree}$ query in one free variable is expressible as an absolute $\mathcal{X}$CPath expression.*

*Digression: Is first order expressivity enough?* [5] argues for the non-counting property of natural languages. A counting property expresses that a path consists of a number of nodes that is a multiple of a given number $k$. Since first order definable properties of trees are non-counting [32], by Theorem 4, $\mathcal{X}$CPath has the non-counting property. Example 2 shows that with regular expressions one can express counting properties. DTD's also allow one to express these: e.g., $A \longrightarrow (B, B, B)^+$ expresses that $A$ nodes have a number of $B$ children divisible by 3.

It seems to us that for the node addressing language first order expressivity is sufficient. This granted, Theorem 4 means that one need not look for other (i.e., more expressive) XPath fragments than $\mathcal{X}$CPath. Whether this also holds for constraints is debatable. Fact is that many natural counting constraints can be equivalently expressed with first order constraints. Take for example, the DTD rule $couple \longrightarrow (man, woman)^*$, which describes the couple element as a sequence of man, woman pairs. Clearly this expresses a counting property, but the same constraint can be expressed by the following $\mathcal{X}$CPath (i.e., first order) inclusions on sets of nodes. Both left and right hand side of the inclusions are filter expressions. (In these rules we just write $man$ instead of the cumbersome self :: $man$, and similarly for the other node labels.)

$$\begin{aligned} couple &\subseteq \mathsf{not} \Downarrow :: *[\mathsf{not}\ man\ \mathsf{and}\ \mathsf{not}\ woman] \\ man\ \mathsf{and} \Uparrow :: couple &\subseteq \mathsf{not} \Rightarrow :: *[\mathsf{not}\ woman] \\ woman\ \mathsf{and} \Uparrow :: couple &\subseteq \Leftarrow :: man\ \mathsf{and}\ \mathsf{not} \Rightarrow :: *[\mathsf{not}\ man]. \end{aligned}$$

In the next two sections on the complexity of query evaluation and containment we will continue discussing more powerful languages than $\mathcal{X}$CPath, partly because there is no difference in complexity, and partly because DTD's are expressible in them.

## 5 Query evaluation

The last two sections are about the computational complexity of two key problems related to XPath: query evaluation and query containment. We briefly discuss the complexity classes used in this paper. For more thorough surveys of the related theory see [20,27]. By PTIME and EXPTIME we denote the well-known complexity classes of problems solvable in deterministic polynomial and deterministic exponential time, respectively, on Turing machines.

$\mathcal{X}_{reg}$ queries can be evaluated in linear time in the size of the data and the query. This is the same bound as for Core XPath. This bound is optimal because the combined complexity of Core XPath is already PTIME hard [16]. It is not hard to see that the linear time algorithm for Core XPath in [15] can be extended to work for full $\mathcal{X}_{reg}$. But the result also follows from known results about Propositional Dynamic Logic model checking [3] by the translation given in Theorem 10. For a model $\mathfrak{M}$, a node $n$ and an XPath expression $q$, $answer_{\mathfrak{M}}^{n}(q) = \{t \mid (n, t) \in [\![q]\!]_{\mathfrak{M}}\}$.

**Theorem 5.** *For $\mathcal{X}_{reg}$ expressions $q$, $answer_{\mathfrak{M}}^{n}(q)$ can be computed in time $O(|\mathfrak{M}| \cdot |q|)$.*

## 6 Query containment under constraints

We discuss equivalence and containment of XPath expressions in the presence of constraints on the document trees. Constraints can take the form of a Document Type Definition (DTD) [37], an XSchema [40] specification, and in general can be any statement. For XPath expressions containing the child and descendant axes and union of paths, this problem —given a DTD— is already EXPTIME-complete [26]. The containment problem has been studied in [36,9,25,26]. For example, consider the XPath expressions in abbreviated syntax[5] $q_1 := a[b]/c$ and $q_2 := a/c$. Then obviously $q_1$ implies $q_2$. But the converse does not hold, as there are trees with $a$ nodes having just a single $c$ child. Of course there are many situations in which the "counterexamples" to the converse containment disappear, and in which $q_1$ and $q_2$ are equivalent. For instance when

(a) every $a$ node has a $b$ child. This is the case when the DTD contains for instance the rule $a \longrightarrow (b, d|e^*)$, or in general on trees satisfying $\mathsf{self} :: a \equiv \mathsf{self} :: a[\Downarrow :: b]$.
(b) if every $c$ node has a $b$ sibling. This holds in all trees in which $\mathsf{self} :: b \equiv \mathsf{self} :: b[\Rightarrow^+ :: c \text{ or } \Leftarrow^+ :: c]$ .
(c) if no $a$ has a $c$ child. This holds in all trees satisfying $\mathsf{self} :: a[\Downarrow :: c] \equiv \emptyset$.

We consider the following decision problem: for $t_i, t_i'$, XPath expressions, does it follow that $t_0 \equiv t_0'$ given that $t_1 \equiv t_1'$ and ... and $t_n \equiv t_n'$ (notation: $t_1 \equiv t_1', \ldots, t_n \equiv t_n' \models t_0 \equiv t_0'$). The example above gives four instances of this problem:

1. $\models a[b]/c \equiv a/c$,
2. $\mathsf{self} :: a \equiv \mathsf{self} :: a[\Downarrow :: b] \models a[b]/c \equiv a/c$,
3. $\mathsf{self} :: b \equiv \mathsf{self} :: b[\Rightarrow^+ :: c \text{ or } \Leftarrow^+ :: c] \models a[b]/c \equiv a/c$,
4. $\mathsf{self} :: a[\Downarrow :: c] \equiv \emptyset \models a[b]/c \equiv a/c$.

This first instance does not hold, all others do. A number of instructive observations can be drawn from these examples. First, the constraints are all expressed as equivalences between XPath expressions denoting *sets of nodes*, while the conclusion relates two sets of *pairs of nodes*. This seems general: constraints on trees are naturally expressed on nodes, not on edges[6]. A DTD is a prime example. From the constraints on sets of nodes we deduce equivalences about sets of pairs of nodes. In example (a) this is immediate by substitution of equivalents. In example (b), some simple algebraic manipulation is needed, for instance using the validity $x[y]/y \equiv x/y$.

That constraints on trees are naturally given in terms of nodes is fortunate because we can reason about sets of nodes instead of sets of edges. The last becomes (on arbitrary graphs) quickly undecidable. The desire for computationally well behaved languages for reasoning on graphs resulted in the development of several languages which can specify sets of nodes of trees or graphs like Propositional Dynamic Logic [19], Computation Tree Logic [11] and the propositional $\mu$-calculus [22]. Such languages

[5] In our syntax they are $q1 := \mathsf{self} :: a[\Downarrow :: b]/\Downarrow :: c$ and $q2 := \mathsf{self} :: a/\Downarrow :: c$.

[6] This does not mean that we cannot express general properties of trees. For instance, $\mathsf{self} :: *[\mathsf{not} \Uparrow :: *] \equiv \mathsf{self} :: *[\mathsf{not} \Downarrow :: */\Downarrow :: */\Downarrow :: *]$ expresses that the tree has depth at most two, and $\mathsf{self} :: *[\mathsf{not} \Uparrow :: *] \equiv \mathsf{self} :: *[\mathsf{not} \Downarrow^* :: *[\mathsf{not} \Downarrow :: */\Rightarrow :: */\Rightarrow :: *]]$ that the tree is at most binary branching.

are –like XPath– typically two-sorted, having a sort for the relations between nodes and a sort for sets of nodes. Concerns about computational complexity together with realistic modeling needs determine which operations are allowed on which sort. For instance, boolean intersection is useful on the set sort (and available in XPath 1.0) but not in the relational sort. Similarly, complementation on the set sort is still manageable computationally but leads to high complexity when allowed on the relation sort.

All these propositional languages contain a construct $\langle\pi\rangle\phi$ with $\pi$ from the edge sort and $\phi$ from the set sort. $\langle\pi\rangle\phi$ denotes the set of nodes from which there *exists* a $\pi$ path to a $\phi$ node. Full boolean operators can be applied to these constructs. Note that XPath contains exactly the same construct: the location path $\pi :: *[\phi]$ evaluated as a filter expression.

Thus we consider the constraint inference problem restricted to XPath expressions denoting sets of nodes, earlier called *filter expressions*, after their use as filters of node sets in XPath. This restriction is natural, computationally still feasible and places us in an established research tradition.

We distinguish three notions of equivalence. The first two are the same as in [4]. The third is new. Two $\mathcal{X}_{reg}$ expressions $p$ and $p'$ are *equivalent* if for every tree model $\mathfrak{M}$, $[\![p]\!]_{\mathfrak{M}} = [\![p']\!]_{\mathfrak{M}}$. They are *root equivalent* if the answer sets of $p$ and $p'$ are the same, when evaluated at the root[7]. The difference between these two notions is easily seen by the following example. The expressions $\mathsf{self} :: *[A \ \mathsf{and\ not}\ A]$ and $\mathsf{self} :: *[\Uparrow]$ are equivalent when evaluated at the root (both denote the empty set) but nowhere else. If $p, p'$ are both filter expressions and $\mathsf{self} :: *[p] \equiv \mathsf{self} :: *[p']$, we call them *filter equivalent*. Equivalence of $p$ and $p'$ is denoted by $p \equiv p'$, letting context decide which notion of equivalence is meant. Root and filter equivalence are closely related notions:

**Theorem 6.** *Root equivalence can effectively be reduced to filter equivalence and vice verse.*

The statement $t_1 \equiv t'_1, \ldots, t_n \equiv t'_n \models t_0 \equiv t'_0$ expresses that $t_0 \equiv t'_0$ is logically implied by the set of constraints $t_1 \equiv t'_1, \ldots, t_n \equiv t'_n$. This is this case if for each model $\mathfrak{M}$ in which $[\![t_i]\!]_{\mathfrak{M}} = [\![t'_i]\!]_{\mathfrak{M}}$ holds for all $i$ between 1 and $n$, also $[\![t_0]\!]_{\mathfrak{M}} = [\![t'_0]\!]_{\mathfrak{M}}$ holds. The following results follow easily from the literature.

**Theorem 7.** *(i) Let $t_0^{(')}, \ldots, t_n^{(')}$ be $\mathcal{X}_{reg}$ expressions.*
*The problem whether $t_1 \equiv t'_1, \ldots, t_n \equiv t'_n \models t_0 \equiv t'_0$ is decidable.*
*(ii) Let $t_0^{(')}, \ldots, t_n^{(')}$ be $\mathcal{X}$Core filter expressions in which `child` is the only axis. The problem whether $t_1 \equiv t'_1, \ldots, t_n \equiv t'_n \models t_0 \equiv t'_0$ is EXPTIME hard.*

Decidability for the problem in (i) is obtained by an interpretation into $S\omega S$, whence the complexity is non-elementary [30]. On the other hand, (ii) shows that a single exponential complexity is virtually unavoidable: it already obtains for filter expressions with the most popular axis. Above we argued that a restriction to filter expressions is a good idea when looking for "low" complexity, and this still yields a very useful fragment. And indeed we have

**Theorem 8.** *Let $t_0^{(')}, \ldots, t_n^{(')}$ be $\mathcal{X}_{reg}$ root or filter expressions. The problem whether $t_1 \equiv t'_1, \ldots, t_n \equiv t'_n \models t_0 \equiv t'_0$ is in EXPTIME.*

[7] Thus, if for every tree model $\mathfrak{M}$, $[\![/p]\!]_{\mathfrak{M}} = [\![/p']\!]_{\mathfrak{M}}$.

## 6.1 Expressing DTDs in $\mathcal{X}_{reg}^{\mathsf{CPath}}$

In this section we show how a DTD can effectively be transformed into a set of constraints on $\mathcal{X}_{reg}^{\mathsf{CPath}}$ filter expressions. The DTD and this set are equivalent in the sense that a tree model conforms to the DTD if and only if each $\mathcal{X}_{reg}^{\mathsf{CPath}}$ filter expression is true at every node. A regular expression is a formula $r$ generated by the following grammar: $r \quad ::= e \mid (r;r) \mid (r \cup r) \mid r^*$ with $e$ an element name. A DTD rule is a statement of the form $e \longrightarrow r$ with $e$ an element name and $r$ a regular expression. A DTD consists of a set of such rules and a rule stating the label of the root. An example of a DTD rule is $family \longrightarrow wife; husband; kid^*$. A document conforms to this rule if each $family$ node has a $wife$, a $husband$ and zero or more $kid$ nodes as children, in that order. But that is equivalent to saying that for this document, the following equivalence holds:

$$\mathsf{self} :: family \equiv \mathsf{self} :: *[\Downarrow :: wife[\mathsf{first\ and} \Rightarrow :: husband[(\Rightarrow_{[kid]})^* :: *[\mathsf{last}]]]],$$

where $\mathsf{first}$ and $\mathsf{last}$ abbreviate $\mathsf{not} \Leftarrow :: *$ and $\mathsf{not} \Rightarrow :: *$, denoting the first and the last child, respectively. This example yields the idea for an effective reduction. Note that instead of $\Rightarrow :: husband[\mathsf{fexpr}]$ we could have used a conditional path: $\Rightarrow_{[husband]} :: *[\mathsf{fexpr}]$. For ease of translation, we assume without loss of generality that each DTD rule is of the form $e \longrightarrow e'; r$, for $e'$ an element name. Replace each element name $a$ in $r$ by $\Rightarrow_{[\mathsf{self} :: a]}$ and call the result $r^t$. Now a DTD rule $e \rightarrow e'; r$ is transformed into the equivalent $\mathcal{X}_{reg}^{\mathsf{CPath}}$ filter expression constraint[8]:

(1) $\mathsf{self} :: e \subseteq \mathsf{self} :: e[\Downarrow :: e'[\mathsf{first\ and}\ r^t :: *[\mathsf{last}]]]$.

That this transformation is correct is most easily seen by thinking of the finite state automaton (FSA) corresponding to $e'; r$. The word to be recognized is the sequence of children of an $e$ node in order. The expression in the right hand side of (1) processes this sequence just like an FSA would. We assumed that every node has at least one child, the first being labeled by $e'$. The transition from the initial state corresponds to making the step to the first child (done by $\Downarrow :: e'[\mathsf{first}]$). The output state of the FSA is encoded by $\mathsf{last}$, indicating the last child. Now $r^t$ describes the transition in the FSA from the first node after the input state to the output state. The expression $\Rightarrow_{[\mathsf{self} :: a]}$ encodes an $a$–labeled transition in the FSA. If a DTD specifies that the root is labeled by $e$, this corresponds to the constraint $root \subseteq self :: e$. (Here and elsewhere $root$ is an abbreviation for $\mathsf{self} :: *[\mathsf{not} \Uparrow :: *]$). So we have shown the following

**Theorem 9.** *Let $\Sigma$ be a DTD and $\Sigma^t$ the set of expressions obtained by the above transformation. Then for each tree $T$ in which each node has a single label it holds that $T$ conforms to the DTD $\Sigma$ iff $T \models \Sigma^t$.*

This yields together with Theorem 8,

**Corollary 1.** *Both root equivalence and filter equivalence of $\mathcal{X}_{reg}$ expressions given a DTD can be decided in* EXPTIME.

[8] The symbol $\subseteq$ denotes set inclusion. Inclusion of node sets is definable as follows: $f_1 \subseteq f_2$ iff $\mathsf{self} :: * \equiv \mathsf{self} :: *[\mathsf{not}\ f_1\ \mathsf{or}\ f_2]$.

## 7 Conclusions

We can conclude that $\mathcal{X}$CPath can be seen as a stable fixed point in the development of XPath languages. $\mathcal{X}$CPath is expressively complete for first order properties on node labeled sibling ordered trees. The extra expressive power comes at no (theoretical) extra cost: query evaluation is in linear and query equivalence given a DTD in exponential time. These results even hold for the much stronger language $\mathcal{X}_{reg}$.

Having a stable fixed point means that new research directions come easily. We mention a few. An important question is whether $\mathcal{X}$CPath is also complete with respect to first order definable paths (i.e., first order formulas in two free variables.) Another expressivity question concerns XPath with regular expressions and tests, our language $\mathcal{X}_{reg}$. Is there a natural extension of first order logic for which $\mathcal{X}_{reg}$ is complete? An empirical question related to first and second order expressivity —see the discussion at the end of Section 4— is whether second order expressivity is really needed, both for XPath and for constraint languages like DTD and XSchema. Finally we would like to have a normal form theorem for $\mathcal{X}_{reg}$ and $\mathcal{X}_{reg}^{\mathsf{CPath}}$. In particular it would be useful to have an effective algorithm transforming $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expressions into directed step expressions (cf. the proof of Theorem 8.)

## References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the web*. Morgan Kaufman, 2000.
2. N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13:1–18, 2003.
3. N. Alechina and N. Immerman. Reachability logic: An efficient fragment of transitive closure logic. *Logic Journal of the IGPL*, 8(3):325–337, 2000.
4. M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. In Proc. ICDT'03, 2003.
5. R. Berwick and A. Weinberg. *The Grammatical Basis of Natural Languages*. MIT Press, Cambridge, MA, 1984.
6. P. Blackburn, B. Gaiffe, and M. Marx. Variable free reasoning on finite trees. In *Proceedings of Mathematics of Language (MOL–8), Bloomington*, 2003.
7. D. Calvanese, G. De Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *J. of Logic and Computation*, 9(3):295–318, 1999.
8. E.M. Clarke and B.-H. Schlingloff. Model checking. Elsevier Science Publishers, to appear.
9. A. Deutsch and V. Tannen. Containment of regular path expressions under integrity constraints. In *Knowledge Representation Meets Databases*, 2001.
10. J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4:405–451, 1970.
11. E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71, Springer, 1981.
12. M. Fisher and R. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.
13. D.M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic*. Oxford Science Publications, 1994. Volume 1: Mathematical Foundations and Computational Aspects.
14. D.M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. 7th ACM Symposium on Principles of Programming Languages*, pages 163–173, 1980.

15. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *Proc. of the 28th International Conference on Very Large Data Bases (VLDB 2002)*, 2002.
16. G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *PODS 2003*, pages 179–190, 2003.
17. G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proc. LICS*, Copenhagen, 2002.
18. D. Harel. Dynamic logic. In D.M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume 2, pages 497–604. Reidel, Dordrecht, 1984.
19. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
20. D. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 67–161. Elsevier, 1990.
21. J.A.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.
22. D. Kozen. Results on the propositional mu-calculus. *Th. Comp. Science*, 27, 1983.
23. D. Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.
24. M. Marx. $\mathcal{X}$CPath, the expressively complete XPath fragment. Manuscript, July 2003.
25. G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. PODS'02*, pages 65–76, 2002.
26. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *ICDT 2003*, 2003.
27. Ch. Papadimitriou. *Computational Complexity*. Addison–Wesley, 1994.
28. V. Pratt. Models of program logics. In *Proceedings FoCS*, pages 115–122, 1979.
29. M. Rabin. Decidability of second order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.
30. K. Reinhardt. The complexity of translating logic to finite automata. In E. Grädel et al., editor, *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*, pages 231–238. 2002.
31. J. Rogers. *A descriptive approach to language theoretic complexity*. CSLI Press, 1998.
32. W. Thomas. Logical aspects in the study of tree languages. In B. Courcelle, editor, *Ninth Colloquium on Trees in Algebra and Programming*, pages 31–50. CUP, 1984.
33. M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.
34. P. Wadler. Two semantics for XPath. Technical report, Bell Labs, 2000.
35. M. Weyer. Decidability of S1S and S2S. In E. Grädel et al., editor, *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*, pages 207–230. Springer, 2002.
36. P. Wood. On the equivalence of XML patterns. In *Proc. 1st Int. Conf. on Computational Logic*, volume 1861 of *LNCS*, pages 1152–1166, 2000.
37. W3C. Extensible markup language (XML) 1.0 `http://www.w3.org/TR/REC-xml`.
38. W3C. XML path language (XPath 1.0). `http://www.w3.org/TR/xpath.html`.
39. W3C. XML path language (XPpath 2.0) `http://www.w3.org/TR/xpath20/`.
40. W3C. XML schema part 1: Structures. `http://www.w3.org/TR/xmlschema-1`.
41. W3C. Xquery 1.0: A query language for XML. `http://www.w3.org/TR//xquery/`.
42. W3C. XSL transformations language XSLT 2.0.`http://www.w3.org/TR/xslt20/`.

## A Appendix

### A.1 XPath and Propositional Dynamic Logic

The next theorem states that PDL and the $\mathcal{X}_{reg}$ filter expressions are equally expressive and effectively reducible to each other.

For our translation it is easier to use a variant of standard PDL in which the state formulas are boolean formulas over formulas $\langle\pi\rangle$, for $\pi$ a program. Now $\langle\pi\rangle$ has the same meaning as $\langle\pi\rangle\top$ in standard PDL. Clearly this variant is equally expressive as PDL by the equivalences $\langle\pi\rangle \equiv \langle\pi\rangle\top$ and $\langle\pi\rangle\phi \equiv \langle\pi; ?\phi\rangle$.

**Theorem 10.** *There are logspace translations $t$ from $\mathcal{X}_{reg}$ filter expressions to PDL formulas and $t'$ in the converse direction such that for all models $\mathfrak{M}$, for all nodes $n$ in $\mathfrak{M}$,*

$$(2) \qquad \mathcal{E}_{\mathfrak{M}}(n, e) = \mathit{true} \iff \mathfrak{M}, n \models (e)^t$$

$$(3) \qquad \mathcal{E}_{\mathfrak{M}}(n, (e^{t'}) = \mathit{true} \iff \mathfrak{M}, n \models e.$$

PROOF OF THEOREM 10. Let $(\cdot)^t$ from $\mathcal{X}_{reg}$ filter expressions to PDL formulas be defined as follows:

$$\begin{array}{ll}
(\text{axis} :: \text{ntst}[\text{fexpr}])^t & = \langle \text{axis}; ?(\text{ntst} \wedge \text{fexpr}^t)\rangle \\
(/\text{locpath})^t & = \langle \Uparrow^*; ?\mathit{root}\rangle(\text{locpath})^t \\
(\text{locpath1}/\text{locpath2})^t & = (\text{locpath1})^t(\text{locpath2})^t \\
(\text{locpath1} \mid \text{locpath2})^t & = (\text{locpath1})^t \vee (\text{locpath2})^t \\
(\cdot)^t \text{ commutes with the booleans} &
\end{array}$$

Then (2) holds. For the other direction, let $(\cdot)^{t'}$ commute with the booleans and let $\langle\pi\rangle^{t'} = \mathsf{self} :: *[\pi^\circ :: *]$, where $\pi^\circ$ is $\pi$ with each occurrence of $?\phi$ replaced by $?\mathsf{self} :: *[\phi^{t'}]$. Then (3) holds.

## A.2 Proofs

PROOF OF THEOREM 2. Every axis containing converses can be rewritten into one without converses by pushing them in with the following equivalences:

$$\begin{array}{lll}
\Downarrow^{-1} \equiv \Uparrow, & (p^{-1})^{-1} \equiv p, & (p/q)^{-1} \equiv q^{-1}/p^{-1}, \\
\Uparrow^{-1} \equiv \Downarrow, & (?p)^{-1} \equiv ?p, & (p \mid q)^{-1} \equiv p^{-1} \mid q^{-1}, \\
\Leftarrow^{-1} \equiv \Rightarrow, & (p^*)^{-1} \equiv (p^{-1})^*, & \\
\Rightarrow^{-1} \equiv \Leftarrow, & & \\
\mathsf{self}^{-1} \equiv \mathsf{self}. & &
\end{array}$$

PROOF OF THEOREM 3. $\mathcal{X}^{\uparrow}_{r,[]}$ is a fragment of Core XPath, but does not have negation on predicates. That explains the first strict inclusion. The definition of $\mathcal{X}\mathsf{Core}$ given here was explicitly designed to make the connection with Core XPath immediate. $\mathcal{X}\mathsf{Core}$ does not have the Core XPath axes `following` and `preceding` as primitives, but they are definable as explained in Remark 1. All other Core XPath axes are clearly in $\mathcal{X}\mathsf{Core}$. The inclusion is strict because Core XPath does not contain the immediate right and left sibling axis.

$\mathcal{X}\mathsf{Core} \subsetneq \mathcal{X}\mathsf{CPath}$ holds already on linear structures. This follows from a fundamental result in temporal logic stating that on such structures the temporal operator *until* is not expressible by the operators *next-time*, *sometime-in-the-future* and their inverses [13]. The last two correspond to the XPath axis `child` and `descendant`, respectively. *Until(A,B)* is expressible with conditional paths as $(\Downarrow_{\mathsf{self}\,::\,B})^* :: *[/]\Downarrow :: A$.

$\mathcal{X}\mathsf{CPath} \subsetneq \mathcal{X}_{reg}$ also holds on linear structures already. $\mathcal{X}\mathsf{CPath}$ is a fragment of the first order logic of ordered trees by Proposition 1. But Example 2 expresses a relation which is not first order expressible on trees [32].

$\mathcal{X}^{\mathsf{CPath}}_{reg} \subseteq \mathcal{X}_{reg}$, because the conditional axis can be expressed by $?$ and $;$. For the other direction, let $p$ be an $\mathcal{X}_{reg}$ axis. Apply to all subterms of $p$ the following rewrite rules until no more is applicable[9]. From the axioms of Kleene algebras with tests [23] it follows that all rules state an equivalence.

$$\begin{array}{lll} (a \cup b); c & \longrightarrow & (a; c) \cup (b; c) \\ c; (a \cup b) & \longrightarrow & (c; a) \cup (c; b) \\ (a^*)^* & \longrightarrow & a^* \\ ?F_1 \cup ?F_2 & \longrightarrow & ?(F_1 \vee F_2) \\ ?F_1; ?F_2 & \longrightarrow & ?(F_1 \wedge F_2) \end{array} \qquad \begin{array}{lll} (?F)^* & \longrightarrow & \mathsf{self} \\ (a \cup ?F)^* & \longrightarrow & a^* \\ a^*; ?F & \longrightarrow & ?F \cup a^*; a; ?F \\ ?F; a^* & \longrightarrow & ?F \cup ?F; a; a^*. \end{array}$$

Then all tests occurring under the scope of a $*$ or in a sequence will be conditional to an atomic axes. Now consider a location step `axes::ntst[fexpr]`. If `axes` is a $*$ expression or a sequence, the location step is in $\mathcal{X}_{reg}^{\mathsf{CPath}}$. If it is a union or a test, delete tests by the equivalences

$$\begin{array}{lll} \texttt{?F::nst[fexpr]} & \equiv & \mathsf{self}\texttt{::nst[fexpr and F]} \\ \texttt{(?F}\cup\texttt{ A)::nst[fexpr]} & \equiv & \texttt{A::nst[fexpr] | }\mathsf{self}\texttt{::nst[fexpr and F]}. \end{array}$$

PROOF OF PROPOSITION 1. The only interesting cases are transitive closures of conditional paths, like $\Downarrow^*_{\text{fexpr}} :: t[E]$. This expression translates to the $\mathcal{L}^{tree}_{FO}$ formula (letting $(\cdot)^\circ$ denote the translation) ; $(x = y \vee \forall z(Descendant(x, z) \wedge Descendant(z, y) \rightarrow \text{fexpr}^\circ(z))) \wedge \text{fexpr}^\circ(y) \wedge t(y) \wedge E^\circ(y)$.

PROOF OF THEOREM 5. Computing the set of states in a model $\mathfrak{M}$ at which a PDL formula $p$ is true can be done in time $O(|\mathfrak{M}| \cdot |p|)$ [3]. Thus for $\mathcal{X}_{reg}$ filter expressions, the result follows from Theorem 10. Now let $q$ be an arbitrary expression, and we want to compute $answer^n_{\mathfrak{M}}(q)$. Expand $\mathfrak{M}$ with a new label $l_n$ such that $[\![l_n]\!]_{\mathfrak{M}} = \{(n, n)\}$. Use the technique in the proof of Theorem 6 to obtain a filter expression $q'$ such that $t \in answer^n_{\mathfrak{M}}(q)$ iff $\mathcal{E}_{\mathfrak{M}}(t, q') = true$ (here use the new label $l_n$ instead of $root$.)

PROOF OF THEOREM 6. We use the fact that $\mathcal{X}_{reg}$ axis are closed under converse (Theorem 2) and that every $\mathcal{X}_{reg}$ location path is equivalent to a simple location path of the form $\text{axis} :: t[\text{fexpr}]$. The last holds as $\text{axis}_1 :: t_1[\text{fexpr}_1]|\text{axis}_2 :: t_2[\text{fexpr}_2]$ is equivalent to $(\text{axis}_1; ?(\mathsf{self} :: t_1[\text{fexpr}_1]) \cup \text{axis}_2; ?(\mathsf{self} :: t_2[\text{fexpr}_2])) :: *$, etc. We claim that for each model $\mathfrak{M}$, $\mathfrak{M} \models /\text{axis}_1 :: t_1[\text{fexpr}_1] \equiv /\text{axis}_2 :: t_2[\text{fexpr}_2]$ if and only if $\mathfrak{M} \models \mathsf{self} :: t_1[\text{fexpr}_1 \text{ and } \text{axis}_1{}^{-1} :: *[root]] \equiv \mathsf{self} :: t_2[\text{fexpr}_2 \text{ and } \text{axis}_2{}^{-1} :: *[root]]$. This is easily proved by writing out the definitions using the equivalence $(p; q)^{-1} \equiv q^{-1}; p^{-1}$. Filter equivalence can be reduced to root equivalence because $p \equiv p'$ iff $\mathsf{self} :: *[root] \equiv \mathsf{self} :: *[root \text{ and not } \Downarrow^* :: *[p \text{ and not } p']]$.

PROOF OF THEOREM 7. (i) Decidability follows from an interpretation in the monadic second order logic over variably branching trees $L^2_{K,P}$ of [31]. The consequence problem for $L^2_{K,P}$ is shown to be decidable by an interpretation into $S\omega S$. Finiteness of a tree can be expressed in $L^2_{K,P}$. The translation of $\mathcal{X}_{reg}$ expressions into $L^2_{K,P}$ is straightforward given the meaning definition. We only have to use second order quantification to define the transitive closure of a relation. This can be done by the usual definition: for $R$ any binary relation, $xR^*y$ holds iff

$$x = y \vee \forall X(X(x) \wedge \forall z, z'(X(z) \wedge zRz' \rightarrow X(z')) \rightarrow X(y)).$$

(ii) Theorem 10 gives an effective reduction from PDL tree formulas to $\mathcal{X}_{reg}$ filter expressions. The consequence problem for ordinary PDL interpreted on graphs is EXPTIME hard [12]. An inspection of the proof shows that the path $\Downarrow$ can be used as the only program and that checking consequence on finite trees is sufficient.

[9] An exponential blowup cannot be avoided. Consider a composition of $n$ expressions $(a_i \cup ?F_i)$, for the $a_i$ atomic axis. Then the only way of rewriting this into an $\mathcal{X}_{reg}^{\mathsf{CPath}}$ axes is to fully distribute the unions over the compositions, leaving a union consisting of $2^n$ elements.

PROOF OF THEOREM 8. We first give a proof for $\mathcal{X}_{reg}$ by a reduction to deterministic PDL with converse. Then we give a direct algorithm which might be easier to implement. This algorithm unfortunately works only for $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expressions in which there are no occurrences of an arrow and its inverse under the Kleene star.

*Proof by reduction.* By Theorem 10, and standard PDL reasoning we need only decide satisfiability of PDL formulas in the signature with the four arrow programs interpreted on finite trees. Call the language compass-PDL. Consider the language $\mathrm{PDL}_2$, the PDL language with only the two programs $\{\downarrow_1, \downarrow_2\}$ and their inverses $\{\uparrow_1, \uparrow_2\}$. $\mathrm{PDL}_2$ is interpreted on finite at most *binary-branching* trees, with $\downarrow_1$ and $\downarrow_2$ interpreted by the first and second daughter relation, respectively. We will effectively reduce compass-PDL satisfiability to $\mathrm{PDL}_2$ satisfiability. $\mathrm{PDL}_2$ is a fragment of deterministic PDL with converse. [33] shows that the satisfiability problem for this latter language is decidable in EXPTIME over the class of all models. This is done by constructing for each formula $\phi$ a tree automaton $A_\phi$ which accepts exactly all tree models in which $\phi$ is satisfied. Thus deciding satisfiability of $\phi$ reduces to checking emptiness of $A_\phi$. The last check can be done in time polynomial in the size of $A_\phi$. As the size of $A_\phi$ is exponential in the length of $\phi$, this yields the exponential time decision procedure.

But we want satisfiability on *finite* trees. This is easy to cope with in an automata-theoretic framework: construct an automaton $A_{\mathit{fin\_tree}}$, which accepts only finite binary trees, and check emptiness of $A_\phi \cap A_{\mathit{fin\_tree}}$. The size of $A_{\mathit{fin\_tree}}$ does not depend on $\phi$, so this problem is still in EXPTIME.

The reduction from compass-PDL to $\mathrm{PDL}_2$ formulas is very simple: replace the compass-PDL programs $\Downarrow, \Uparrow, \Rightarrow, \Leftarrow$ by the $\mathrm{PDL}_2$ programs $\Downarrow_1;\Downarrow_2^*, \Uparrow_2^*;\Uparrow_1, \Downarrow_2, \Uparrow_2$, respectively. It is straightforward to prove that this reduction preserves satisfiability, following the reduction from $S\omega S$ to $S2S$ as explained in [35]: a compass-PDL model $(T, R_\Rightarrow, R_\Downarrow, V)$ is turned into an $\mathrm{PDL}_2$ model $(T, R_1, R_2, V)$ by defining $R_1 = \{(x, y) \mid xR_\Downarrow y$ and $y$ is the first daughter of $x\}$ and $R_2 = R_\Rightarrow$. A $\mathrm{PDL}_2$ model $(T, R_1, R_2, V)$ is turned into a compass-PDL model $(T, R_\Rightarrow, R_\Downarrow, V)$ by defining $R_\Rightarrow = R_2$ and $R_\Downarrow = R_1 \circ R_2^*$.

*Direct proof.* Consider $t_1 \equiv t_1', \ldots, t_n \equiv t_n' \Rightarrow t_0 \equiv t_0'$ as in the Theorem. By Theorem 6 we may assume that all terms are filter expressions. For this proof, assume also that the expressions are in $\mathcal{X}_{reg}^{\mathsf{CPath}}$ and that in each subexpression $\pi^*$ of an axis, $\pi$ may not contain an arrow and its inverse. We call such expressions *directed*. First we bring these expressions into a normal form. Define the set of *step paths* by the grammar $s ::= a \mid {}_{[\mathtt{test}]}a \mid a_{[\mathtt{test}]} \mid s \cup \ldots \cup s \mid p_1; \ldots ; p_n; s; p_{n+1}; \ldots ; p_{n+k}$, where $a$ is one of the four arrows and the $p_i$ can be any path. The next lemma gives the reason for considering directed step paths. Its simple proof is omitted.

**Lemma 1.** *Let $\mathcal{X}$ be a step path not containing an arrow axis and its inverse. Then in any model $\mathfrak{M}$, $n\mathcal{X}n'$ holds only if $n \neq n'$ and the relation $\mathcal{X}$ is conversely well-founded.*

**Lemma 2.** *Every $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expression is effectively reducible to an $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expression in which for every occurrence of $\pi^*$, $\pi$ is a step path.*

*Proof.* Define a function $G$ from $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expressions to sets of $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expressions which yields the set of step paths which “generate” the expression: for an $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expression $p$, $G(p) = \{p\}$, if $p$ is a step path. Otherwise set $G(p \cup q) = G(p) \cup G(q)$, $G(p;q) = G(p) \cup G(q)$ and $G(p^*) = G(p)$. Now define the translation $(\cdot)^t$ from $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expressions to $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expressions in which for every occurrence of $\pi^*$, $\pi$ is a step path: $p^t = p$, if $p$ is a step path. Otherwise set $(p \cup q)^t = p^t \cup q^t$, $(p;q)^t = p^t; q^t$ and $(p^*)^t = (s_1 \cup \ldots \cup s_k)^*$ for $G(p) = \{s_1, \ldots, s_k\}$. A rather straightforward induction shows that $p^t \equiv p$ and that $|p^t|$ is linear in $|p|$.

The rest of the proof is a reduction to the consequence problem for a simple language interpreted on binary trees. The language is equivalent to filter expressions with only two axis, first and second child. We could give the proof in terms of the $\mathcal{X}_{reg}^{\mathsf{CPath}}$ language, but the syntax is very cumbersome to work with. For that reason we use the effective reduction to PDL formulas from Theorem 10 and do the rest of the proof in the PDL syntax.

The proof consists of two linear reductions and a decision algorithm. The first reduction removes the transitive closure operation by adding new propositional symbols. Similar techniques are employed in [29,10] for obtaining normalized monadic second order formulas. The second reduction is based on Rabin's reduction of $S\omega S$ to $S2S$.

Let $\mathcal{L}_2$ be the modal language[10] with only two atomic paths $\{\downarrow_1, \downarrow_2\}$ and the modal constant $root$. $\mathcal{L}_2$ is interpreted on finite *binary* trees, with $\downarrow_1$ and $\downarrow_2$ interpreted by the first and second daughter relation, respectively, and $root$ holds exactly at the root. The semantics is the same as that of PDL.

The *consequence problem* for PDL and for $\mathcal{L}_2$ is the following: for $\phi_0, \ldots, \phi_n$ determine whether $\phi_1, \ldots, \phi_n \models \phi_0$ is true. The last is true if for each model $\mathfrak{M}$ with node set $T$ in which $[\![\phi_1]\!]_{\mathfrak{M}} = \ldots [\![\phi_n]\!]_{\mathfrak{M}} = T$ it also holds that $[\![\phi_0]\!]_{\mathfrak{M}} = T$.

**Lemma 3.** *The $\mathcal{L}_2$ consequence problem is decidable in* EXPTIME.

*Proof.* A direct proof of this lemma using a bottom up version of Pratt's [28] EXPTIME Hintikka Set elimination technique was given in [6]. As $\mathcal{L}_2$ is a sublanguage of $\mathrm{PDL}_2$, the lemma also follows from the argument given above in the *proof by reduction*.

A PDL formula in which the programs are $\mathcal{X}_{reg}^{\mathsf{CPath}}$ axis with the restriction that for every occurrence of $\pi^*$, $\pi$ is a step path not containing an arrow and its inverse is called a *directed step PDL formula*. The next Lemma together with Lemmas 2 and 3 and Theorem 10 yield the EXPTIME result for directed $\mathcal{X}_{reg}^{\mathsf{CPath}}$ expressions.

**Lemma 4.** *There is an effective reduction from the consequence problem for directed step PDL formulas to the consequence problem for $\mathcal{L}_2$ formulas.*

*Proof.* Let $\chi$ be a directed step PDL formula. Let $Cl(\chi)$ be the Fisher–Ladner closure [12] of $\chi$. We associate a formula $\nabla(\chi)$ with $\chi$ as follows. We create for each $\phi \in Cl(\chi)$, a new propositional variable $q_\phi$ and for each $\langle p_1; p_2\rangle\phi \in Cl(\chi)$ we also create $q_{\langle p_1\rangle q_{\langle p_2\rangle\phi}}$. Now $\nabla(\chi)$ "axiomatizes" these new variables as follows:

$$\begin{array}{lll}
q_p & \leftrightarrow & p \\
q_{\neg\phi} & \leftrightarrow & \neg q_\phi \\
q_{\phi\wedge\psi} & \leftrightarrow & q_\phi \wedge q_\psi \\
q_{\phi\vee\psi} & \leftrightarrow & q_\phi \vee q_\psi \\
q_{\langle\alpha\rangle\phi} & \leftrightarrow & \langle\alpha\rangle q_\phi \\
q_{\langle \mathtt{test}\,\alpha\rangle\phi} & \leftrightarrow & q_{\mathtt{test}} \wedge \langle\alpha\rangle q_\phi \\
q_{\langle\alpha_{\mathtt{test}}\rangle\phi} & \leftrightarrow & \langle\alpha\rangle(q_{\mathtt{test}} \wedge q_\phi)
\end{array}
\qquad
\begin{array}{lll}
q_{\langle\pi_1;\pi_2\rangle\phi} & \leftrightarrow & q_{\langle\pi_1\rangle q_{\langle\pi_2\rangle\phi}} \\
q_{\langle\pi_1\cup\pi_2\rangle\phi} & \leftrightarrow & q_{\langle\pi_1\rangle\phi} \vee q_{\langle\pi_2\rangle\phi} \\
q_{\langle\pi^*\rangle\phi} & \leftrightarrow & q_\phi \vee q_{\langle\pi\rangle q_{\langle\pi^*\rangle\phi}}
\end{array}$$

for $\alpha \in \{\Downarrow, \Uparrow, \Leftarrow, \Rightarrow\}$, and `test` a directed step PDL formula.

We claim that for every model $\mathfrak{M}$ which validates $\nabla(\chi)$, for every node $n$ and for every new variable $q_\phi$, $[\![q_\phi]\!]_{\mathfrak{M}} = [\![\phi]\!]_{\mathfrak{M}}$. In the proof we use the usual logical notation $n \models \phi$ instead of the more cumbersome $n \in [\![\phi]\!]_{\mathfrak{M}}$.

The proof is by induction on the structure of the formula and the path and for the left to right direction of the $\langle\pi^*\rangle$ case by induction on the depth of direction of $\pi$. The case for proposition

[10] That is, the syntax is defined as that of PDL, with $\pi ::= \downarrow_1 | \downarrow_2$ and $\phi ::= l \mid \top \mid root \mid \neg\phi \mid \phi\wedge\phi \mid \phi\vee\phi \mid \langle\pi\rangle\phi$.

letters and the Boolean cases are immediate by the axioms in $\nabla(\chi)$. For the diamonds, we also need an induction on the structure of the paths, so let $n \models q_{\langle\pi\rangle\phi}$ for $\pi \in \{\Downarrow, \Uparrow, \Leftarrow, \Rightarrow\}$. Iff, by the axiom, $n \models \langle\pi\rangle q_\phi$. Iff $m \models q_\phi$ for some $m$ such that $nR_\pi m$. Iff, by induction hypothesis, $m \models \phi$ for some $m$ such that $nR_\pi m$. Iff $n \models \langle\pi\rangle\phi$. The cases for the conditional arrows are shown similarly. The cases for composition and union and the right to left direction of the Kleene star case follow immediately from the axioms. For the left to right direction of the Kleene star we also do an induction on the depth of the direction of $\pi$. This is possible by Lemma 1. As an example let the direction be $\Downarrow$. Let $n$ be a leaf and let $n \models q_{\langle\pi^*\rangle\phi}$. By the axiom in $\nabla(\chi)$, $n \models q_\phi$ or $n \models q_{\langle\pi\rangle q_{\langle\pi^*\rangle\phi}}$. The latter implies, by induction on the complexity of paths, that $n \models \langle\pi\rangle q_{\langle\pi^*\rangle\phi}$. But that is not possible by Lemma 1 as $n$ is a leaf. Thus $n \models q_\phi$, and by IH, $n \models \phi$, whence $n \models \langle\pi^*\rangle\phi$. Now let $n$ be a node with $k+1$ descendants, and let the claim hold for nodes with $k$ descendants. Let $n \models q_{\langle\pi^*\rangle\phi}$. Then by the axiom $n \models q_\phi$ or $n \models q_{\langle\pi\rangle q_{\langle\pi^*\rangle\phi}}$. In the first case, $n \models \langle\pi^*\rangle\phi$ by IH. In the second case, by the induction on the structure of paths, $n \models \langle\pi\rangle q_{\langle\pi^*\rangle\phi}$. As $\pi$ is a directed step path, there exists a child $m$ of $n$ and $m \models q_{\langle\pi^*\rangle\phi}$. Whence, by the induction on the depth of the direction $\mathfrak{M}, m \models \langle\pi^*\rangle\phi$. But then also $\mathfrak{M}, n \models \langle\pi^*\rangle\phi$.

Hence for $\phi_0, \phi_1, \dots, \phi_n$ directed step PDL formulas, we have

$$\phi_1, \dots, \phi_n \models \phi_0 \iff q_{\phi_1}, \dots, q_{\phi_n}, \nabla(\{\phi_0, \phi_1, \dots, \phi_n\}) \models q_{\phi_0}.$$

As the language is closed under conjunction, we need only consider problems of the form $\phi_1 \models \phi_0$, and we do that from now on.

Note that the only modalities occurring in $\nabla(\chi)$ are $\langle\pi\rangle$ for $\pi$ one of the four arrows. We can further reduce the number of arrows to only $\Downarrow, \Rightarrow$ when we add two modal constants *root* and *first* for the root and first elements, respectively. Let $\chi$ be a formula in this fragment. As before create a new variable $q_\phi$ for each (single negation of a) subformula $\phi$ of $\chi$. Create $\nabla(\chi)$ as follows: $q_p \leftrightarrow p$, $q_{\neg\phi} \leftrightarrow \neg q_\phi$, $q_{\phi\wedge\psi} \leftrightarrow q_\phi \wedge q_\psi$ and $q_{\langle\pi\rangle\phi} \leftrightarrow \langle\pi\rangle q_\phi$, for $\pi \in \{\Downarrow, \Rightarrow\}$, and for each subformula $\langle\Uparrow\rangle\phi$ and $\langle\Leftarrow\rangle\phi$, add to $\nabla\chi$ the axioms

$$\begin{array}{lll} q_\phi \to [\Downarrow]q_{\langle\Uparrow\rangle\phi}, & \langle\Downarrow\rangle q_{\langle\Uparrow\rangle\phi} \to q_\phi, & q_{\langle\Uparrow\rangle\phi} \to \neg root, \\ q_\phi \to [\Rightarrow]q_{\langle\Leftarrow\rangle\phi}, & \langle\Rightarrow\rangle q_{\langle\Leftarrow\rangle\phi} \to q_\phi, & q_{\langle\Leftarrow\rangle\phi} \to \neg first. \end{array}$$

We claim that for every model $\mathfrak{M}$ which validates $\nabla(\chi)$, for every node $n$ and for every subformula $\phi \in Cl(\chi)$, $\mathfrak{M}, n \models q_\phi$ iff $\mathfrak{M}, n \models \phi$. An easy induction shows this. Consider the case of $\langle\Uparrow\rangle\phi$. If $n \models \langle\Uparrow\rangle\phi$, then the parent of $n$ models $\phi$, whence by inductive hypothesis, it models $q_\phi$, so by the axiom $q_\phi \to [\Downarrow]q_{\langle\Uparrow\rangle\phi}$, $n \Vdash q_{\langle\Uparrow\rangle\phi}$. Conversely, if $n \models q_{\langle\Uparrow\rangle\phi}$, then by axiom $q_{\langle\Uparrow\rangle\phi} \to \neg root$, $n$ is not the root. So the parent of $n$ exists and it models $\langle\Downarrow\rangle q_{\langle\Uparrow\rangle\phi}$. Then it models $q_\phi$ by axiom $\langle\Downarrow\rangle q_{\langle\Uparrow\rangle\phi} \to q_\phi$, and by inductive hypothesis it models $\phi$. Thus $n \models \langle\Uparrow\rangle\phi$. Hence, $\gamma \models \chi \iff \nabla(\gamma \wedge \chi), q_\gamma \models q_\chi$.
Note that the formulas on the right hand side only contain the modalities $\langle\Downarrow\rangle$ and $\langle\Rightarrow\rangle$. Now we come to the second reduction: to the consequence problem of binary branching trees. Let $\chi$ be a formula, translate it to a $\mathrm{PDL}_2$ formula as in the *proof by reduction*. Note that this is a directed step PDL formula. Finally use the first reduction again to reduce the consequence problem to the consequence problem of the language with just the modalities $\langle\Downarrow_1\rangle$ and $\langle\Downarrow_2\rangle$, interpreted on binary trees.

Clinching all these reductions together, we obtain the reduction stated in Lemma 4.