Conditional XPath, the first order complete XPath dialect*

Maarten Marx†
Language and Inference Technology
University of Amsterdam
marx@science.uva.nl

ABSTRACT

XPath is the W3C–standard node addressing language for XML documents. XPath is still under development and its technical aspects are intensively studied. What is missing at present is a clear characterization of the expressive power of XPath, be it either semantical or with reference to some well established existing (logical) formalism. Core XPath (the logical core of XPath 1.0 defined by Gottlob et al.) cannot express queries with conditional paths as exemplified by "do a `child` step, while `test` is true at the resulting node." In a first-order complete extension of Core XPath, such queries are expressible. We add *conditional* axis relations to Core XPath and show that the resulting language, called conditional XPath, is equally expressive as first-order logic when interpreted on ordered trees. Both the result, the extended XPath language, and the proof are closely related to temporal logic. Specifically, while Core XPath may be viewed as a simple temporal logic, conditional XPath extends this with (counterparts of) the since and until operators.

1. INTRODUCTION

XPath 1.0 [25] is a variable free language used for selecting nodes from XML documents. XPath plays a crucial role in other XML technologies such as XSLT [29], XQuery [28] and XML schema constraints, e.g., [27]. The recent XPath 2.0 [26] is much more expressive and is close to being a full fledged tree query language. It contains variables which are used in if–then–else, for, and quantified expressions. The available axis relations are the same in both versions. As XPath 2.0 adds variables and first order quantifiers it is natural to ask whether XPath 1.0 is already expressively complete with respect to first order logic. More precisely, is every first order query $\phi(x)$ which selects a set of nodes from an XML document model equivalent to an XPath expression?

As we will see, it is not, but a simple and natural addition is sufficient. This paper introduces and motivates this addition, makes the connection with temporal logic and shows expressive completeness of the resulting language.

We use the abstraction to the logical core of XPath 1.0 (called *Core XPath)* developed in [9, 8]. Core XPath is interpreted on XML document tree models. The central expression in XPath is the location path $\mathsf{axis} :: \mathit{node\_label}[\mathsf{filter}]$ 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'.

Core XPath cannot express the "until–like" query whose answer set consists of all nodes n' satisfying the first order statement "n' is a descendant of n, the label of n' is A, and for all z, if z is a descendant of n and n' is a descendant of z, then the label of z is A [15]. This query can be expressed by $(\texttt{child} :: A)^+$, which we call a *conditional axis*. As usual, R^+ denotes the transitive closure of the (binary) relation R. 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. We show that XPath with the additional conditional axis relations, abbreviated as CXPath, is closely related to temporal logic and use this connection to establish the expressive completeness result.

Both the result, the new XPath language and the proof can be seen as direct offspring of the seminal POPL 1980 paper by Gabbay, Pnueli, Shelah and Stavi [6]. There it was shown that the propositional temporal language with since and until (now known as LTL) is just as expressive as first order logic over linear structures. This result, which generalized and gave an accessible proof to Kamp's Theorem [12], is arguably the clear technical basis which led to the acceptance and widespread use of temporal logic in computer science. XPath is already present in most XML applications. The expressive completeness result of this paper justifies this central position, and establishes CXPath as a natural fixed point

†Research supported by NWO grant 612.000.106.

in the development of the family of XPath languages.

While CXPath is more expressive than Core XPath, in all other aspects it is very conservative. CXPath has the same unabbreviated syntax as XPath 1.0 and Core XPath, the same (standard W3C) semantics, the same (linear time) upper bound for query evaluation, and the same (exponential time) upper bound for query containment given a DTD or a set of XSchema constraints [15].

The paper is organized as follows. Section 2 defines the XPath language we study, gives examples and motivates our main result. Section 3 makes the connection with first order and temporal logic. Expressive completeness is shown in Section 4, and complexity is discussed in Section 5. Proofs of all claims in the text stated without proof are given in the appendix.

Related work.

Variable free logical languages for reasoning about (finite) sibling ordered trees exists since the early nineties [2, 14, 18]. These were developed as alternatives to monadic second order logic which was used to develop model theoretic syntax [20]. The study of expressive completeness with respect to first order logic of modal languages originated with Kamp's thesis [12]. He showed that LTL is expressively complete with respect to Dedekind complete linear structures. [6] made this result accessible and generalized it. Gabbay's notion of separation yielded a proof understandable to the non-specialist; Stavi's until for gaps gave completeness with respect to all linear structures. Immerman and Kozen [11] showed that first order logic with at most k variables is expressively complete for k–branching trees. A model–theoretic version of the separation property is Shelah's Composition Method [23].

Schlingloff [21] generalized Kamp's Theorem to bounded and unbounded trees. Besides since and until, he has a connective counting the number of daughters making a wff true:

$$t \models \mathcal{X}_k A \iff \exists t_1 \dots t_k \bigwedge_{1 \le i \neg j \le k} (tR_{\Downarrow} t_i \wedge t_i \neq t_j \wedge t_i \models A).$$

Note that $\mathcal{X}_k$ is expressible over ordered trees. For k bounded trees, the language with since and until plus all $\mathcal{X}_i$, for $2 \le i \le k$ is expressively complete. For unbounded trees, $\mathcal{X}_i$ is needed for every natural number i. Unfortunately this result does not imply separation.[1]

Expressive completeness for a language closely related to ours was announced in [18]. Unfortunately, [18] does not mention any of the above works, the proof does not use separation, it is over 50 pages long and very hard to follow. Here we just mentioned work about *first order* expressive completeness of variable free languages. [19] is a recent survey on the expressivity of temporal logics covering also second order expressivity results.

[1]This is most easily explained using the notation introduced below. Consider the wff $\langle\Uparrow\rangle\langle\Downarrow\rangle A$. On linear structures, this is equivalent to $\langle\Uparrow\rangle\top \wedge A$, on ordered trees to $\langle\Uparrow\rangle\top \wedge (A \vee \langle\Leftarrow^+\rangle A \vee \langle\Rightarrow^+\rangle A)$, which are both separated formulas. But the signature of unordered trees is too weak to separate this formula.

2. A BRIEF INTRODUCTION TO XPATH

[8] proposes a fragment of XPath 1.0 which can be seen as its logical core, but lacks much of the functionality that account for little expressive power. In effect it supports all XPath's axis relations, except the attribute relation[2], it allows sequencing and taking unions of path expressions and full booleans in the filter expressions. It is called Core XPath, also referred to as *navigational XPath*. A similar logical abstraction is made in [1]. As the focus of this paper is expressive power, we also restrict XPath to its logical core.

We will define two 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 [8]. The expressions obey the standard W3C unabbreviated XPath 1.0 syntax, except for the different notation of the axis relations. The semantics is as in [1] and [7], which is in line with the standard XPath semantics from [30].

Our simplest language $\mathcal{X}_{Core}$ is slightly more expressive than Core XPath (cf. Remark 2). We view $\mathcal{X}_{Core}$ as the baseline in expressive power for XPath languages. CXPath, for *conditional* XPath, simply extends $\mathcal{X}_{Core}$ with conditional paths.

DEFINITION 1. The syntax of the XPath languages $\mathcal{X}_{Core}$ and CXPath is defined by the grammar

locpath	::=	axis '::'ntst | axis '::' ntst '['fexpr']' | '/' locpath | locpath '/' locpath | locpath '|' locpath
fexpr	::=	locpath | `not` fexpr | fexpr `and` fexpr | fexpr `or` fexpr
axis	::=	self | primitive_axis | primitive_axis$^+$ | primitive_axis*.

The primitive axis of $\mathcal{X}_{Core}$ are $\Downarrow$, $\Uparrow$, $\Rightarrow$ and $\Leftarrow$, and those of CXPath are

$$| \Downarrow | \Uparrow | \Rightarrow | \Leftarrow | \Downarrow_{\text{fexpr}} | \Uparrow_{\text{fexpr}} | \Rightarrow_{\text{fexpr}} | \Leftarrow_{\text{fexpr}}.$$

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

[2]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.

[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

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

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

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

Table 1: The semantics of $\mathcal{X}_{Core}$ and CXPath.

with a set of primitive symbols 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 fexpr is most naturally viewed as denoting a set of nodes: all n such that $\mathcal{E}(n, \mathsf{fexpr})$ is true. For examples, we refer to below and to [8]. Given a tree $\mathfrak{M}$ and an expression A, the denotation or meaning of A in $\mathfrak{M}$ is written as $[\![A]\!]_{\mathfrak{M}}$. Table 1 contains the definition of $[\![\cdot]\!]_{\mathfrak{M}}$.

Examples

Consider the following information need: *give elements whose next element in document order has tag A*. This can be expressed in first order logic by

$$\exists y(x \ll y \wedge A(y) \wedge \neg\exists z(x \ll z \ll y)),$$

in which $\ll$ abbreviates `descendant` or `ancestor_or_self`/`following_sibling`/`descendant_or_self`. To "program" this information need in navigational XPath we seem to need to express the "next in document order" relation. To do this we use a few macros: $\mathsf{first}, \mathsf{last}$ and leaf abbreviate $\mathsf{self} :: *[\texttt{not} \Rightarrow :: *]$, $\mathsf{self} :: *[\texttt{not} \Leftarrow :: *]$ and $\mathsf{self} :: *[\texttt{not} \Downarrow :: *]$, respectively. We also use the converses[4] of the conditional axis, written as ${}_{\mathsf{last}}\Uparrow^+$ and so on. The meaning of ${}_{\mathsf{last}}\Uparrow^+$ is the transitive closure of the relation $\mathsf{self} :: *[\mathsf{last}]/\Uparrow :: *$.

Now we can write the "next in document order" relation in conditional XPath by the following case distinction:

$$\Downarrow :: *[\mathsf{first}] \mid \mathsf{self} :: *[\mathsf{leaf}]/\Rightarrow :: * \mid {}_{\mathsf{last}}\Uparrow^+ :: */\Rightarrow :: *.$$

Having this it is easy to express the information need.

Our second example is an exercise suggested by Michael Benedikt. Here the information need is in essence a Boolean query. We implement that by returning the root of the document if and only if the Boolean query evaluates to true. The example is *check whether nodetags of the frontier of the tree (that is, the set of leaves in document order) is a word in $(ab)^*$*. It is not very hard to write a first order sentence capturing this query. The crucial relation is the *next frontier node* relation, expressible as

$$\mathsf{self} :: *[\mathsf{leaf}]/N/\mathsf{self} :: *[\mathsf{leaf}],$$

where N is the union of $\Rightarrow :: *$ and

$${}_{\mathsf{last}}\Uparrow^+ :: */\Rightarrow :: */\Downarrow^*_{\mathsf{first}} :: *.$$

The rest of the exercise is fairly easy as all dependencies are local.

Motivation

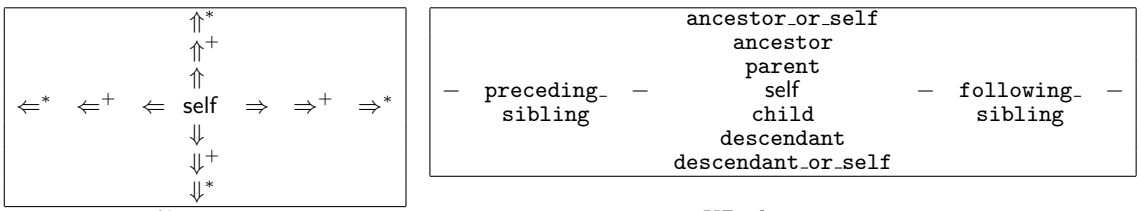
Taking the risk of advocating the obvious, we give a brief motivation of the main result of this paper. Consider the two examples just given. From the natural language description it is not obvious that one can express them at all in navigational XPath. Though also not immediate, it is a lot easier to express the information needs in first order logic, that is, giving a formal specification of the problem. The expressive completeness theorem can be viewed as an insurance policy: once a formal specification of the desired set of nodes in first order logic is given, one is insured that there exists an XPath "program" which computes the set. Of course it can, and

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 $n' = s \cdot k + 1$.

[4]Unlike $\mathcal{X}_{Core}$, the axis relations of CXPath are not closed under taking converses. Still ${}_{\phi}\Uparrow^+ :: t[\psi]$ is definable as

$$\mathsf{self} :: *[\phi]/\Uparrow^*_{\phi} :: */\Uparrow :: t[\psi],$$

and similarly for the other directions.



$\mathcal{X}_{Core}$ axis

XPath 1.0 axis

Figure 1: Primitive axis of $\mathcal{X}_{Core}$ and XPath 1.0.

often will, be hard to *find* an equivalent XPath expression but it is guaranteed to exist. This insurance policy puts the original problem in a radically different light. Compare this with giving a student an exercise which you know is possible to solve (though you might not know the solution) and an exercise for which you do not know it. In the second case, when will you stop encouraging or pushing the student?

Pushing the analogy a bit further, one could say: "Well you gave me an insurance policy, can't you give me an insurance agent which *compiles* my first order specification into an equivalent XPath expression?". We can, but the practical value will be rather limited in general, as we cannot give an elementary time bound for such a compiler (see Section 5).

REMARK 2. XPath 1.0 (and hence Core XPath) has a peculiar 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 13 axis as primitives. They are given on the left hand side of Figure 1. The right hand side contains the corresponding 9 primitive "compass" axis of XPath 1.0. XPath 1.0 also has two primitive axis related to the document order. These are just syntactic sugar, as witnessed by the following definitions:

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

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

3. CONDITIONAL XPATH, FIRST ORDER AND TEMPORAL LOGIC

We now view XPath as a query language over trees and compare it to first order and temporal logic interpreted on trees. Before we can start, we must make clear what kind of queries XPath expresses. This is formalized by the notion of the answer set of an XPath expression. The answer set of `locpath` evaluated on a tree $\mathfrak{M}$ (notation: $answer_{\mathfrak{M}}(\texttt{locpath})$) is the set

$$\{n \in \mathfrak{M} \mid \text{there exists an } m,\ (m,n) \in [\![\texttt{locpath}]\!]_{\mathfrak{M}}\}.$$

Thus for each expression A, $answer_{\mathfrak{M}}(A)$ equals

$$answer_{\mathfrak{M}}(/\Downarrow^* :: */A).$$

An XPath expression starting with /, indicating that it should be evaluated at the root, is called an *absolute* expression. We will show that for every first order query $\phi(x)$ there exists an absolute CXPath expression A which is equivalent in the following strong sense: for each tree $\mathfrak{M}$, for each node n, $\mathfrak{M} \models \phi(n)$ if and only if $n \in answer_{\mathfrak{M}}(A)$.

Conditional XPath and First order logic

Let $\mathcal{L}_{FO}^{tree}$ be the first order language in the signature with two binary relation symbols $<$ and $\prec$ and countably many unary predicates $P, Q, \ldots$ $\mathcal{L}_{FO}^{tree}$ is interpreted on node labeled sibling ordered trees in the obvious manner: $<$ is interpreted as the descendant relation $R_{\Downarrow}^{+}$, $\prec$ as the strict total order $R_{\Rightarrow}^{+}$ on the siblings, and the unary predicates P as the sets of nodes labeled with P.

One might expect that XPath expressions correspond to conjunctive queries, but this is not true due to the free use of boolean operators inside the filter expressions. It is easy to see that the $\mathcal{X}_{Core}$ expressions are equivalent to $\mathcal{L}_{FO}^{tree}$ formulas. A little bit harder is

PROPOSITION 3. *Every* CXPath *(filter) expression is, on ordered trees, equivalent to an $\mathcal{L}_{FO}^{tree}$ formula in two (one) free variable(s). Moreover, the $\mathcal{L}_{FO}^{tree}$ formula contains at most three variables.*

The converse of this proposition would state that CXPath 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 [12]. The main result of the present paper is a generalization to ordered trees:

THEOREM 4. *Every $\mathcal{L}_{FO}^{tree}$ formula in one free variable is, on ordered trees, equivalent to an* CXPath *filter expression.*

Note that we do not make a restriction to *finite* trees. The result holds for the class of all trees. For each filter expression `fexpr`, $\mathcal{E}_{\mathfrak{M}}(n, \texttt{fexpr})$ is true if and only if $n \in answer_{\mathfrak{M}}(/\Downarrow^* :: *[\texttt{fexpr}])$. Thus we have

COROLLARY 5. *Every $\mathcal{L}^{tree}_{FO}$ formula in one free variable is, on ordered trees, equivalent to an absolute* CXPath *expression.*

Both Proposition 3 and Theorem 4 have a constructive proof by translation. The easy direction has a linear time translation, the hard direction a translation which takes necessarily non–elementary time (see Section 5). The proof (given in the following section) makes a detour via temporal logic.

Conditional XPath and temporal logic.

Several authors observed that fragments of XPath can be embedded into Computation Tree Logic CTL [17, 9]. Temporal logic comes with operators which look toward the future and toward the past. Interpreted on trees, these operators obtain their meaning from the descendant relation and its inverse. Because we work on *sibling ordered* trees, it is natural to add temporal operators which work in the horizontal or sibling dimension as well. This is exactly what we do. Define $\mathcal{X}_{until}$ as the propositional modal language with four binary until–like modal operators $\Downarrow, \Uparrow, \Rightarrow, \Leftarrow$. The syntax is given by the grammar

$$\phi ::= p_i \mid \top \mid \neg\phi \mid \phi \wedge \phi \mid \pi(\phi, \phi),$$

with $i \in \omega$ and $\pi \in \{\Downarrow, \Uparrow, \Rightarrow, \Leftarrow\}$. The p_i are propositional variables. The semantics is the standard one for temporal logic, with each arrow interpreted as a strict "until" over the relation corresponding to the direction of the arrow. Formally, a model $\mathfrak{M}$ is a structure (T, h), with T a tree and h an assignment function from the set of propositional variables to the powerset of the set of tree nodes. Truth of a formula is defined relative to a model $\mathfrak{M}$ and a node n in that model via the following recursive definition: $\mathfrak{M}, n \models p_i$ iff $n \in h(p_i)$; $\mathfrak{M}, n \models \neg\phi$ iff $\mathfrak{M}, n \not\models \phi$; $\mathfrak{M}, n \models \phi \wedge \psi$ iff $\mathfrak{M}, n \models \phi$ and $\mathfrak{M}, n \models \psi$; $\mathfrak{M}, t \models \pi(\phi, \psi)$ iff there exists a t' such that $tR^+_\pi t'$ and $\mathfrak{M}, t' \models \phi$ and for all t'' such that $tR^+_\pi t'' R^+_\pi t'$ it holds that $\mathfrak{M}, t'' \models \psi$.

PROPOSITION 6. *Every $\mathcal{X}_{until}$ formula is, on ordered trees, equivalent to an* CXPath *filter expression.*

The converse also holds, as a direct corollary of Propositions 3, 6 and the expressive completeness of $\mathcal{X}_{until}$, to be proved in the next section.

4. EXPRESSIVE COMPLETENESS OF CONDITIONAL XPATH

This section contains the proof of Theorem 4. Instead of proving that Theorem directly we show expressive completeness of the "temporal language" $\mathcal{X}_{until}$, which is sufficient by Proposition 6. We say that $\mathcal{X}_{until}$ is expressively complete if for every $\mathcal{L}^{tree}_{FO}$ formula $\phi(x)$ there exists an $\mathcal{X}_{until}$ formula θ such that for every tree model $\mathfrak{M}$, for all nodes n in $\mathfrak{M}$, $\mathfrak{M} \models \phi(n)$ if and only if $\mathfrak{M}, n \models \theta$. The change of perspective to temporal logic allows us to use directly the techniques developed in [6]. The proof given here follows as closely as possible the very clear presentation in Section 10.2 of [5]. This involves some change of terminology and notation from the previous sections (e.g., instead of formula we use wff, propositional variables are called atoms, etc). The key idea of the proof is the brilliant notion of *separation.*

4.1 Separation

Let $(T, R_\Downarrow, R_\Rightarrow)$ be a tree and $t \in T$. Define the following partition on T:

$$\begin{array}{lcl} present(t) & = & \{t\} \\ future(t) & = & \{s \mid tR^+_\Downarrow s\} \\ past(t) & = & \{s \mid t(R^+_\Uparrow \cup R^+_\Uparrow \circ (R^+_\Leftarrow \cup R^+_\Rightarrow) \circ R^*_\Downarrow)s\} \\ left(t) & = & \{s \mid tR^+_\Leftarrow \circ R^*_\Downarrow s\} \\ right(t) & = & \{s \mid tR^+_\Rightarrow \circ R^*_\Downarrow s\}. \end{array}$$

Note that this partition is different from the one in the XPath 1.0 specification given by the axis relations `self` for present, `descendant` for future, `ancestor` for past, and `following` and `preceding` for right and left, respectively. Our "past" is a much larger set defined by the relation

$$\texttt{ancestor} \circ (\mathsf{self} \cup \texttt{following} \cup \texttt{preceding}).$$

For lack of a better name, we simply called it "past".

Now let h, h' be two assignments and $t \in T$. We say that h, h' *agree on the future of* t iff for any atom q and any $s \in future(t)$, $s \in h(q)$ iff $s \in h'(q)$. We similarly define this notion for the present, past, left and right.

We say that a wff A is a *pure future wff* iff for each tree $\mathfrak{T}$, for all $t \in T$, for all assignments h, h', if h, h' *agree on the future of* t, then $t, h \models A$ iff $t, h' \models A$. Similarly, we define pure present, past, left and right wffs.

We say that a wff A is *separable* iff there exists a wff which is a boolean combination of pure present, future, past, left and right wffs and is equivalent to A everywhere on any tree.

THEOREM 7. *If every $\mathcal{X}_{until}$ wff is separable over trees, then $\mathcal{X}_{until}$ is expressively complete.*

The proof can be copied from the proof of Theorem 9.3.1 in [5] which considers linear flows of time. The only change is to use the partition in five sets given above instead of the past, present and future for linear time.

THEOREM 8. *Each $\mathcal{X}_{until}$ wff is, over trees, separable.*

The proof is provided in the next subsection. As an immediate corollary we obtain expressive completeness of $\mathcal{X}_{until}$.

The next lemma describes a syntactic criterion for pure wffs. Theorem 8 is shown by rewriting each wff into a boolean combination of wffs satisfying these syntactic criteria. For π one of the four orientations, a π wff is a wff whose main connective is $\pi(\cdot, \cdot)$.

LEMMA 9.
1. Each boolean combination of atoms is a pure present wff.
2. Each boolean combination of $\Downarrow$ wffs in whose scope occur only atoms, $\Downarrow$, $\Leftarrow$ and $\Rightarrow$ wffs is a pure future wff.
3. Each boolean combination of $\Rightarrow$ wffs in whose scope occur only atoms, $\Rightarrow$ wffs and pure future wffs is a pure right wff.

4. Each boolean combination of $\Leftarrow$ wffs in whose scope occur only atoms, $\Leftarrow$ wffs and pure future wffs is a pure left wff.
5. Each boolean combination of $\Uparrow$ wffs in whose scope occur only atoms, $\Uparrow$ wffs and pure left and right wffs is a pure past wff.

4.2 Separating formulas

Now we prove that each $\mathcal{X}_{until}$ formula is separable over ordered trees. The proof follows the same structure as the one given in Section 10.2 of [5] for integer time. Note that integer time (or rather, natural number time) is a special case of ordered trees in which each $\Rightarrow$ and $\Leftarrow$ wff is equivalent to $\bot$.

We shall describe a syntactic procedure for separating each wff into a boolean combination of wffs of the form described in Lemma 9. Then that Lemma yields the theorem. Before we start let us summarize what we need to do. We must

1. pull out $\Leftarrow$ wffs from under the scope of $\Rightarrow$ wffs, and conversely;
2. pull out $\Uparrow$ wffs from under the scope of $\Leftarrow, \Rightarrow$ and $\Downarrow$ wffs;
3. pull out $\Downarrow$ wffs from under the scope of $\Uparrow$ wffs.

It should be clear that if we can manage this, the result is in the form described in Lemma 9. 1) follows directly from the linear case, as well as pulling out $\Uparrow$ from under $\Downarrow$, as the past of a tree is a linear structure. Pulling $\Uparrow$ out from under $\Leftarrow$ and $\Rightarrow$ is easy. The real work is in showing 3).

With the first lemma we bring each formula into a normal form.

LEMMA 10. The following are valid on all structures, for π any of the four orientations $\{\Uparrow, \Downarrow, \Leftarrow, \Rightarrow\}$:

$$\begin{aligned} \pi(A \vee B, C) &\equiv \pi(A, C) \vee \pi(B, C) \\ \pi(A, B \wedge C) &\equiv \pi(A, B) \wedge \pi(A, C). \end{aligned}$$

Let a *literal* be an atom or its negation or (a negation of) an orientation wff $\pi(C, D)$ where C is a *conjunction*, and D a *disjunction* of literals. By boolean reasoning and Lemma 10 each wff is equivalent to a wff constructed from literals using conjunction and disjunction only. So we only have to focus attention on literals.

NOTATION 11. As usual in temporal logic, it is convenient to create macros for the unary temporal connectives. For π one of the four arrows, $\langle\pi\rangle A$ abbreviates $\pi(A, \bot)$ and $\langle\pi^+\rangle A$ abbreviates $\pi(A, \top)$. $[\pi]A$ abbreviates $\neg\langle\pi\rangle\neg A$. $\langle\pi\rangle$ corresponds to the "next" operator, and $\langle\pi^+\rangle$ to its transitive closure. $[\Downarrow^+]A$ expresses that everywhere below A holds.

Gabbay's separation version of Kamp's theorem allows us to separate $\Rightarrow$ and $\Leftarrow$ wffs. Call a wff *horizontal* if it does not contain $\Downarrow$ and $\Uparrow$ wffs.

THEOREM 12 (GABBAY). *Each horizontal wff is equivalent to a wff in which no $\Rightarrow$ wff occurs in the scope of a $\Leftarrow$ wff and conversely.*

The next lemma allows us to bring $\Uparrow$ wffs out of the scope of horizontal wffs.

LEMMA 13. Let a, q, A, B be arbitrary wffs. The following are valid over trees. They are also valid if $\Leftarrow$ is replaced everywhere by $\Rightarrow$.

1. $\Leftarrow(a \wedge \Uparrow(A, B), q) \equiv \Leftarrow(a, q) \wedge \Uparrow(A, B)$;
2. $\Leftarrow(a \wedge \neg\Uparrow(A, B), q) \equiv \Leftarrow(a, q) \wedge \neg\Uparrow(A, B)$;
3. $\Leftarrow(a, q \vee \Uparrow(A, B)) \equiv \Leftarrow(a, q) \vee (\Uparrow(A, B) \wedge \langle\Leftarrow^+\rangle a)$;
4. $\Leftarrow(a, q \vee \neg\Uparrow(A, B)) \equiv \Leftarrow(a, q) \vee (\neg\Uparrow(A, B) \wedge \langle\Leftarrow^+\rangle a)$.

We are halfway through the proof. As pure future and pure past formulas may contain pure left and right formulas (Lemma 9), the only cases left are $\Uparrow$ wffs in the scope of $\Downarrow$ wffs and conversely. The next Lemma, which is a copy of Lemma 10.2.3 in [5] adjusted to trees takes care of these. Note that the equivalents are pure past and pure future wffs. Thus the Lemma tells us that we can separate all these formulas. The proof is given in the Appendix.

LEMMA 14. Let a, q, A and B be atoms. Consider the followings wffs:

1. $\Uparrow(a \wedge \Downarrow(A, B), q)$,
2. $\Uparrow(a \wedge \neg\Downarrow(A, B), q)$,
3. $\Uparrow(a, q \vee \Downarrow(A, B))$,
4. $\Uparrow(a, q \vee \neg\Downarrow(A, B))$,
5. $\Uparrow(a \wedge \Downarrow(A, B), q \vee \Downarrow(A, B))$,
6. $\Uparrow(a \wedge \neg\Downarrow(A, B), q \vee \Downarrow(A, B))$,
7. $\Uparrow(a \wedge \Downarrow(A, B), q \vee \neg\Downarrow(A, B))$,
8. $\Uparrow(a \wedge \neg\Downarrow(A, B), q \vee \neg\Downarrow(A, B))$.

(i) Each of the above wffs is equivalent, over ordered trees, to another wff in which the only appearances of the $\Downarrow$ connective are as $\Downarrow(A, B)$ and if an appearance of that wff is in the scope of $\Uparrow$, then $\Downarrow(A, B)$ is in the scope of a $\Leftarrow$ or a $\Rightarrow$ wff (which itself is in the scope of the $\Uparrow$).

(ii) Exchange $\Uparrow$ and $\Downarrow$ in the above wffs. For those wffs, each of them is equivalent, over ordered trees, to another wff in which the only appearances of the $\Uparrow$ connective are as $\Uparrow(A, B)$ and no appearance of that wff is in the scope of $\Downarrow$. Moreover the wff does not contain any $\Leftarrow$ or $\Rightarrow$ subformulas.

The final step.

We now know the basic steps of the separation proof. We simply keep pulling out $\Downarrow$'s from under the scope of $\Uparrow$'s etcetera until there are no more. Given a wff A, this process will eventually lead to a syntactically separated wff, i.e. a wff B which is a boolean combination of wffs as occurring in the left hand side of Lemma 9. Clearly then, by that Lemma, B is separated.

This process of pulling out goes by an inductive process exactly the same as in Lemmas 10.2.4 to 10.2.8 in [5]. With this we finish the proof of Theorem 8.

5. COMPLEXITY

Query evaluation for Core XPath is hard for PTIME (combined complexity) and can be done in time $O(|D| \cdot |Q|)$, with $|D|$ the size of the data and $|Q|$ the size of the query [7, 8]. CXPath is more expressive, but the upper bound remains [15]. Because we have expressive completeness, it is interesting to compare this to results for first order logic. Query evaluation for first order queries is PSPACE complete and can be done in time $O(|D|^n \cdot |Q|)$, where n is the number of variables in Q [3, 10, 24]. So we can explain PTIME–completeness of CXPath by the fact that it is a subset of $\mathcal{L}_{FO}^{tree}$ with at most three variables (Proposition 3). That it is undesirable to allow an unlimited number of variables in queries (whence to compute tables of unlimited size) becomes even clearer when we look at parametrized complexity. [4] showed that under some mild assumptions there is no model checking algorithm for $\mathcal{L}_{FO}^{tree}$ on the class of unbounded trees whose running time is bounded by $f(|Q|) \cdot p(|D|)$, for an elementary function f and a polynomial p.

Satisfiability for conditional XPath over finite trees is complete for exponential time [15]. Satisfiability of first order sentences with $<$ over finite words is in no elementary space-bounded complexity class. Whence, no translation from CXPath into the first order logic of trees can be elementary space-bounded.

6. CONCLUSION

We defined an easy to use, variable free XPath dialect which is expressively complete with respect to first order logic when interpreted on ordered trees. XPath is already present in most XML applications. The expressive completeness result of this paper justifies this central position, and establishes CXPath as a natural fixed point in the development of the family of XPath languages. We think that the lack of variables is one of the reasons for the success of XPath, so it is nice to know that they are not needed for expressivity reasons. Besides that, query evaluation for CXPath can still be done in linear time, while for first order logic there is no algorithm which is elementary in the query and polynomial in the data [4].

Several research questions remain. Is CXPath closed under intersection and complementation, as defined in [1]? Is CXPath also expressively complete with respect to first order formulas in *two free variables*? We conjecture that all these questions can be answered positively. Neven and Schwentick have argued that not FO but unary MSO should be the goal of a natural XPath dialect. Several proposals have been made (we mention the *efficient tree logic* of Neven and Schwentick [16] and monadic datalog of Gottlob and Koch [9]) but, as far as we know, none have a simple variable free XPath like syntax. In [15] we considered next to conditional XPath also a language called *regular* XPath. It has almost the same syntax as conditional XPath, but now also transitive closure of arbitrary location paths is allowed. E.g., the non first order query $(\Downarrow :: */\Downarrow :: *)^+$ is expressible in it. Is (some natural extension of) regular XPath expressively complete for unary MSO? At present we only know that one of the simplest truly MSO queries (though not a unary but a boolean query) –the boolean circuit problem– can be expressed in regular XPath (see [16] for a formulation of the problem in terms of trees).

7. REFERENCES

[1] M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. In *Proc. ICDT'03*, 2003.

[2] P. Blackburn and W. Meyer-Viol. Linguistics, logic, and finite trees. *Logic J. of the IGPL*, 2:3–29, 1994.

[3] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of 9th ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[4] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proc. LICS'02*, pages 215–224, 2002.

[5] D.M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic*. Oxford Science Publications, 1994. Volume 1: Mathematical Foundations and Computational Aspects.

[6] 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.

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

[8] G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *Proc. PODS 2003*, pages 179–190, 2003.

[9] G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proc. LICS'02*, 2002.

[10] N. Immerman. Upper and lower bounds for first order expressibility. *J. Comput. Syst. Sci.*, 25:76–98, 1982.

[11] N. Immerman and D. Kozen. Definability with bounded number of bound variables. In *Proc. LICS* , pages 236–244, Washington, 1987. Computer Society Press.

[12] J.A.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, U. of California, LA, 1968.

[13] C. Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach. In *VLDB 2003*, pages 249–260, 2003.

[14] M. Kracht. Inessential features. In Christian Retore, editor, *Logical Aspects of Computational Linguistics*, number 1328 in LNAI, pages 43–62. Springer, 1997.

[15] M. Marx. XPath with conditional axis relations. In *Proc. EDBT'04*, pages 477–494. 2004.

[16] F. Neven and T. Schwentick. Expressive and Efficient Pattern Languages for Tree-Structured Data. In *Proc. PODS'02*, pages 145–156. 2000.

[17] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. PODS'02*, pages 65–76, 2002.

[18] A. Palm. *Transforming tree constraints into formal grammars*. PhD thesis, Universität Passau, 1997.

[19] A. Rabinovich. Expressive power of temporal logics. In *CONCUR 2002, Proceedings*, volume 2421 of *LNAI*, pages 57–75, 2002.

[20] J. Rogers. *A descriptive approach to language theoretic complexity.* CSLI Press, 1998.

[21] B-H. Schlingloff. Expressive completeness of temporal logic of trees. *Journal of Applied Non–Classical Logics*, 2(2):157–180, 1992.

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

[23] W. Thomas. Ehrenfeucht Games, the Composition Method, and the Monadic Theory of Ordinal Words. In *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht.* Pages 118–143,Springer, 1997.

[24] M. Vardi. On the complexity of bounded–variable queries. In *Proceedings PODS-95*, pages 266–276, 1995.

[25] W3C. XML path language (XPath): Version 1.0. `http://www.w3.org/TR/xpath.html`.

[26] W3C. XML path language (XPath): Version 2.0. `http://www.w3.org/TR/xpath20/`.

[27] W3C. XML schema part 1: Structures. `http://www.w3.org/TR/xmlschema-1`.

[28] W3C. Xquery 1.0: A query language for XML. `http://www.w3.org/TR//xquery/`.

[29] W3C. XSL transformations language (XSLT): Version 2.0. `http://www.w3.org/TR/xslt20/`.

[30] P. Wadler. Two semantics for XPath. Technical report, Bell Labs, 2000.

APPENDIX

PROOF OF PROPOSITION 3. We provide a translation $(\cdot)^t$ from CXPath expressions into the language of relation algebra[5] in the signature $\{<, \prec, P_0, P_1, \dots\}$. This is easier and yields the stronger result, as every relation algebraic expression is equivalent to a first order formula in two free variables and with at most three variables [22].

First we create some abbreviations: $R_{\Downarrow} \equiv\ < \cap \overline{(< \circ <)}$, $R_{\Rightarrow} \equiv \prec \cap \overline{(\prec \circ \prec)}$, and for every expression A, $?A \equiv (1' \cap A)$.

The only tricky part in the translation is that of the transitive closure of a conditional axis. For example, $(n, m) \in [\![(\Downarrow_p)^+]\!]_{\mathfrak{M}}$ iff the test p succeeds at m, $n < m$ and there is no n' with $n < n' < m$ at which the test p fails. This translates very naturally to $< \circ ?p \cap \overline{(< \circ ?\overline{p} \circ <)}$.

The translation $(\cdot)^t$ uses two other translations $(\cdot)^{tl}$ and $(\cdot)^{tf}$ translating labels and location paths occuring as filter expressions, respectively. It is given in Table 2. By just writing out the definitions we obtain that for every CXPath expression A, for every tree $\mathfrak{M}$, and for all nodes x, y, $(x, y) \in [\![A]\!]_{\mathfrak{M}}$ if and only if $\mathfrak{M} \models xA^t y$. Moreover, each filter expression A is translated as $?A^{tf}$, which is equivalent to $1' \cap A^{tf}$, which translates to a first order formula in one free variable. QED

[5]We use $\cap$, $\cup$ and $\overline{(\cdot)}$ for the boolean operators, and $\circ$ and $(\cdot)^{-1}$ for composition and inverse, respectively. $1'$ denotes the identity relation and $\top$ the universal relation. In relation algebra (not to be confused with relational algebra) every expression denotes a *binary* relation.

$$
\begin{array}{lcl}
(\mathsf{self} :: n[\phi])^t & = & ?(n^{tl} \cap \phi^{tf}) \\
(\Downarrow :: n[\phi])^t & = & R_{\Downarrow} \circ ?(n^{tl} \cap \phi^{tf}) \\
(\Rightarrow :: n[\phi])^t & = & R_{\Rightarrow} \circ ?(n^{tl} \cap \phi^{tf}) \\
(\Uparrow :: n[\phi])^t & = & R_{\Downarrow}^{-1} \circ ?(n^{tl} \cap \phi^{tf}) \\
(\Leftarrow :: n[\phi])^t & = & R_{\Rightarrow}^{-1} \circ ?(n^{tl} \cap \phi^{tf}) \\
(\pi_{\psi} :: n[\phi])^t & = & (\pi :: n[\phi])^t \circ ?\psi^{tf} \\
(\Downarrow^+ :: n[\phi])^t & = & < \circ ?(n^{tl} \cap \phi^{tf}) \\
(\Downarrow^* :: n[\phi])^t & = & (\mathsf{self} :: n[\phi])^t \cup (\Downarrow^+ :: n[\phi])^t \\
((\Downarrow_{\psi})^+ :: n[\phi])^t & = & \overline{(< \circ ?\psi^{tf}) \cap (< \circ ?\overline{\psi^{tf}} \circ <)} \circ ?(n^{tl} \cap \phi^{tf}) \\
((\Downarrow_{\psi})^* :: n[\phi])^t & = & (\mathsf{self} :: n[\phi])^t \cup ((\Downarrow_{\psi})^+ :: n[\phi])^t
\end{array}
$$

and similarly for the other three arrows.

$$
\begin{array}{lcl}
(/\mathtt{locpath})^t & = & (1' \cup <^{-1}) \circ ?\overline{(<^{-1} \circ \top)} \circ (\mathtt{locpath})^t \\
(\mathtt{locpath}_1/\mathtt{locpath}_2)^t & = & \mathtt{locpath}_1{}^t \circ \mathtt{locpath}_2{}^t \\
(\mathtt{locpath}_1 \mid \mathtt{locpath}_2)^t & = & \mathtt{locpath}_1{}^t \cup \mathtt{locpath}_2{}^t.
\end{array}
$$

$$
\begin{array}{lcl}
n^{tl} & = & \begin{cases} \top & \text{if } n = * \\ n & \text{otherwise} \end{cases} \\
(\mathtt{locpath})^{tf} & = & (\mathtt{locpath})^t \circ \top
\end{array}
$$

$(\cdot)^{tf}$ commutes with the booleans.

Table 2: Translating CXPath into relation algebra.

PROOF OF PROPOSITION 6. Consider the translation $(\cdot)^t$ from $\mathcal{X}_{until}$ formulas to CXPath filter expressions:

$$
\begin{array}{lcl}
(p_i)^t & = & \mathsf{self} :: p_i \\
(\neg\phi)^t & = & \mathsf{not}\ \phi^t \\
(\phi \wedge \psi)^t & = & \phi^t\ \mathsf{and}\ \psi^t \\
(Until_{\pi}(\phi, \psi))^t & = & \pi :: *[\phi^t]\ \mathsf{or}\ (\pi_{\psi^t})^+ :: */\pi :: *[\phi^t].
\end{array}
$$

Writing out the definitions it is immediate that for every $\mathcal{X}_{until}$ formula ϕ, for every tree $\mathfrak{M}$, and for every node n, it holds that $\mathfrak{M}, n \models \phi$ if and only if $\mathcal{E}(n, \phi^t)$ is true. QED

PROOF OF LEMMA 9. Simple. QED

PROOF OF LEMMA 10. Simple. QED

PROOF OF LEMMA 13. The equivalences follow from the observation that if $xR_{\Rightarrow}^+ y$, then for any $\Uparrow(A, B)$, $x \models \Uparrow(A, B)$ if and only if $y \models \Uparrow(A, B)$. QED

PROOF OF LEMMA 14. Part (ii) follows immediately from Lemma 10.2.3 in [5] because the past of a tree is a linear structure. For part (i), we follow the argumentation in that Lemma, but adjust it where needed.

1. Let $t \models \Uparrow(a \wedge \Downarrow(A, B), q)$ and $s \models a \wedge \Downarrow(A, B)$. Let the node which forces A be u. Thus $sR_{\Downarrow}^+ t$ and $sR_{\Downarrow}^+ u$. In a tree, there are 5 excluding cases for such a situation:
   - $t = u$
   - $tR_{\Downarrow}^+ u$
   - $uR_{\Downarrow}^+ t$
   - $t \neq u$ and not $tR_{\Downarrow}^+ u$ and not $uR_{\Downarrow}^+ t$ and $\forall z(sR_{\Downarrow}^+ zR_{\Downarrow}^+ u \rightarrow \neg zR_{\Downarrow}^* t)$

- $t \neq u$ and not $tR^+_{\Downarrow}u$ and not $uR^+_{\Downarrow}t$ and $\exists z(sR^+_{\Downarrow}zR^+_{\Downarrow}u \wedge zR^+_{\Downarrow}t)$.

The first three separate in the same way as in linear structures. For the last two, we use a formula which will be used repeatedly. It states "my parent forces $\Downarrow(A,B)$ because of a sibling of mine". This can be expressed by a disjunction of pure left and pure right wffs as:

$$\langle\Leftarrow^+\rangle(A \vee (B \wedge \Downarrow(A,B))) \vee \langle\Rightarrow^+\rangle(A \vee (B \wedge \Downarrow(A,B))). \quad (1)$$

Now cases 4 and 5 split each into two subcases:

- $\forall z(sR^+_{\Downarrow}zR^+_{\Downarrow}u \rightarrow \neg zR^*_{\Downarrow}t)$ and ($sR_{\Downarrow}t$ or $\exists w(sR^+_{\Downarrow}wR^+_{\Downarrow}t)$)
- $\exists z(sR^+_{\Downarrow}zR^+_{\Downarrow}u \wedge zR^+_{\Downarrow}t)$ and ($zR_{\Downarrow}t$ or $\exists w(zR^+_{\Downarrow}wR^+_{\Downarrow}t)$).

All these cases together correspond to the following separation of $\Uparrow(a \wedge \Downarrow(A,B), q)$:

$$\begin{array}{ll} & \Uparrow(a, q \wedge B) \wedge A \\ \vee & \Uparrow(a, q \wedge B) \wedge B \wedge \Downarrow(A,B) \\ \vee & \Uparrow(A \wedge q \wedge \Uparrow(a, q \wedge B), q) \\ \vee & \langle\Uparrow\rangle a \wedge (1) \\ \vee & \Uparrow(q \wedge \langle\Uparrow\rangle a \wedge (1), q) \\ \vee & \langle\Uparrow\rangle(q \wedge B \wedge \Uparrow(a, q \wedge B)) \wedge (1) \\ \vee & \Uparrow(q \wedge \Uparrow(a, q \wedge B) \wedge (1), q). \end{array}$$

2. Let $t \models \Uparrow(a \wedge \neg\Downarrow(A,B), q)$ and $s \models a \wedge \neg\Downarrow(A,B)$. Consider two excluding cases:
   - $\forall z(sR^+_{\Downarrow}zR^+_{\Downarrow}t \rightarrow z \models B)$.
   - $\exists z(sR^+_{\Downarrow}zR^+_{\Downarrow}t \wedge z \models \neg B)$.

   In the first case, all z such that $sR^+_{\Downarrow}zR^*_{\Downarrow}t$ force $\neg A \wedge \neg(1)$. In addition, t also forces $\neg(B \vee \Downarrow(A,B))$. In this case, $\Uparrow(a \wedge \neg\Downarrow(A,B), q)$ is equivalent to $\Uparrow(a, q \wedge B \wedge \neg A \wedge \neg(1)) \wedge \neg A \wedge \neg(1) \wedge \neg(B \vee \Downarrow(A,B))$.
   For the second case, let z be the first such $\neg B$ node below s. Then z forces

   $$\neg A \wedge \neg B \wedge q \wedge \Uparrow(a, B \wedge q \wedge \neg A \wedge \neg(1)). \quad (2)$$

   We consider two subcases: $zR_{\Downarrow}t$ and $\exists w : zR^+_{\Downarrow}wR^+_{\Downarrow}t$. In these cases, $\Uparrow(a \wedge \neg\Downarrow(A,B), q)$ is equivalent to $\langle\Uparrow\rangle(2) \wedge \neg(1) \wedge \neg A$ (abbreviate this wff by $(*)$) and $\Uparrow(q \wedge (*), q)$, respectively.
   Summing up $\Uparrow(a \wedge \neg\Downarrow(A,B), q)$ is equivalent to

   $$\begin{array}{lll} \Uparrow(a, q \wedge B \wedge \neg A \wedge \neg(1)) \wedge \neg A \wedge \neg(1) \wedge \neg(B \vee \Downarrow(A,B)) & & \\ \vee \quad \Uparrow((2), \bot) \wedge \neg(1) \wedge \neg A & (*) & \\ \vee \quad \Uparrow(q \wedge (*), q). & & \end{array}$$

3. To separate $\Uparrow(a, q \vee \Downarrow(A,B))$ we look at its negation and use the validity (cf. Lemma 10.2.2 in [5])

   $$\neg\Uparrow(A,B) \equiv [\Uparrow]\neg A \vee \Uparrow(\neg A \wedge \neg B, \neg A). \quad (3)$$

   to obtain $\Uparrow(a, q \vee \Downarrow(A,B)) \equiv \neg([\Uparrow]\neg a \vee \Uparrow(\neg a \wedge \neg q \wedge \neg\Downarrow(A,B), \neg a))$. The right hand side can now be separated with elimination (2).

4. $\Uparrow(a, q \vee \neg\Downarrow(A,B))$ can be separated on linear time into

   $$\begin{array}{ll} & \Uparrow(a, \neg a \wedge [\Uparrow(\neg q \wedge \neg a, \neg a \wedge B) \rightarrow \neg A]) \\ \wedge & (\Uparrow(\neg q \wedge \neg a, \neg a \wedge B) \rightarrow \neg A \wedge \neg(B \vee \Downarrow(A,B))). \end{array}$$

   To obtain the equivalent on branching structures we should add formulas for the cases that $\Downarrow(A,B)$ is forced because of another branch. This is simply done by adding to both occurances of $\neg A$ in the above formula the conjunct $\neg(1)$.

5. $\Uparrow(a \wedge \Downarrow(A,B), q \vee \Downarrow(A,B))$ can be separated on linear time[6] into

   $$\begin{array}{ll} & \Uparrow(a, B) \wedge (A \vee (B \vee \Downarrow(A,B))) \\ \vee & \Uparrow(A \wedge \Uparrow(a, B), \Uparrow(\neg q, \neg A) \rightarrow A \vee B) \\ \wedge & \Uparrow(\neg q, \neg A) \rightarrow A \vee (B \vee \Downarrow(A,B)). \end{array}$$

   The first disjunct holds when the A from $\Downarrow(A,B)$ is true in the future or present of the node of evalutaion, the second when it is true in the past. As in the previous case we must add formulas in the second disjunct allowing for $\Downarrow(A,B)$ being forced in another branch. This is done by simply adding (1) as a disjunct to the consequent of the implications in the second and third line

6. The case of $\Uparrow(a \wedge \neg\Downarrow(A,B), q \vee \Downarrow(A,B))$. Let s be the node forcing $a \wedge \neg\Downarrow(A,B)$. On linear time the formula separates into two disjuncts, considering when the first occurrence (if any) of $\neg B$ after s is:

   $$\begin{array}{ll} & \Uparrow(a, q \wedge \neg A) \wedge \neg A \wedge \neg(B \wedge \Downarrow(A,B)) \\ \vee & \Uparrow(\neg B \wedge \neg A \wedge (q \vee \Downarrow(A,B)) \wedge \Uparrow(a, q \wedge \neg A), \\ & q \vee \Downarrow(A,B). \end{array}$$

   As in the previous two cases, we obtain the equivalent formula on branching structures by considering the branches: add (1) as a disjunct to each occurance of $\neg A$ in the above formula. Eliminations (3) and (5) can be used to finish the separating.

7. Be prepared, this is again a seven case distinction. Let $t \models \Uparrow(a \wedge \Downarrow(A,B), q \vee \neg\Downarrow(A,B))$ and $s \models a \wedge \Downarrow(A,B)$. Let the node which forces A be u. Hence $sR^+_{\Downarrow}t$ and $sR^+_{\Downarrow}u$. Consider the same 5 cases as in elimination (1). The last two cases split each into the same two subcases as in elimination (1).
   All these seven cases together correspond to the formula in Figure 2. This formula is not separated yet. The third disjunct can be separated by using distribution in the first argument and then eliminations (4) and (8). The fifth and the last are separated by elimination (4).

8. The last case is exactly the same as with linear structures. We include the argument for completeness of the paper. The case $D = \Uparrow(a \wedge \neg\Downarrow(A,B), q \vee \neg\Downarrow(A,B))$ can be reduced to cases already discussed since

   $$\begin{array}{lll} \neg\Uparrow(a \wedge x, q \vee y) & \equiv & [\Uparrow](\neg a \vee \neg x) \\ & \vee & \Uparrow(\neg q \wedge \neg y \wedge \neg a, \neg a \vee \neg x) \\ & \vee & \Uparrow(\neg q \wedge \neg y \wedge \neg x, \neg a \vee \neg x). \end{array}$$

   Substituting $y = x = \neg\Downarrow(A,B)$ we obtain

   $$\begin{array}{lll} \neg D & \equiv & [\Uparrow](\neg a \vee \Downarrow(A,B)) \\ & \vee & \Uparrow(\neg q \wedge \Downarrow(A,B) \wedge \neg a, \neg a \vee \Downarrow(A,B)) \\ & \vee & \Uparrow(\neg q \wedge \Downarrow(A,B) \wedge \Downarrow(A,B), \neg a \vee \Downarrow(A,B)). \end{array}$$

   Notice that the last disjunct in $\neg D$ is redundant. These cases can be handled by other eliminations, especially (5). QED

[6] For this case, the formula in [5] contains a typo. The last conjunction in the last line should be a disjunction.

$$
\begin{array}{lcl}
 & \Uparrow(a, B \wedge q) \wedge A & \\
\vee & \Uparrow(a, B \wedge q) \wedge B \wedge \Downarrow(A, B) & \\
\vee & \Uparrow(A \wedge \Uparrow(a, B \wedge q) \wedge (q \vee \neg\Downarrow(A, B)), q \vee \neg\Downarrow(A, B)) & \\
\vee & \langle\Uparrow\rangle(a \wedge (1)) & \text{(call this wff (*))} \\
\vee & \Uparrow((*), q \vee \neg\Downarrow(A, B)) & \\
\vee & \langle\Uparrow\rangle(\Uparrow(a, B \wedge q) \wedge B \wedge q) \wedge (1) & \text{(call this wff (**))} \\
\vee & \Uparrow((**) \wedge B \wedge q, q \vee \neg\Downarrow(A, B)). &
\end{array}
$$

Figure 2: The formula of case 7.