Efficient and Expressive Tree Filters

Michael Benedikt[1] and Alan Jeffrey[2]

[1] Computing Laboratory, Oxford University
[2] Bell Labs, Alcatel-Lucent

Abstract. We investigate streaming evaluation of filters on XML documents, evaluated both at the root node and at an arbitrary node. Motivated by applications in protocol processing, we are interested in algorithms that make one pass over the input, using space that is independent of the data and polynomial in the filter. We deal with a logic equivalent to the XPath language, and also an extension with an Until operator. We introduce restricted sublanguages based on looking only at "reversed" axes, and show that these allow polynomial space streaming implementations. We further show that these fragments are expressively complete. Our results make use of techniques developed for the study of Linear Temporal Logic, applied to XML filtering.

1 Introduction

The eXtensible Markup Language (XML) is a common standard for data exchange on the Web. In a common scenario an application is required to manipulate an incoming XML document online, processing it as a stream of tags, using limited memory. This can occur in XML-based subscription services: an application registers for one or more XML feeds, and filters from within these the XML data that is of interest. A very different sort of application is in monitoring XML-based protocols; here the goal is to determine of the data as a whole (that is, the protocol message) whether it should be forwarded for further processing. What both scenarios have in common is the need for a flexible filtering description mechanism and a stream processor that can enforce these filter descriptions.

In terms of the description mechanism, the typical assumption is that filtering will be specified in some variant of the XPath language [25]. In this work we will look at *filters* defined in several languages:

- HML, a logic equivalent in expressiveness to *Navigational XPath* – the fragment of XPath in which only the tag structure of the document is utilized, ignoring the attribute and PCDATA content.
- +HML, a fragment of HML which is equivalent in expressiveness to Positive XPath, the subset of Navigational XPath without negation.
- $\mathcal{X}_{\text{until}}$, an extension of HML equivalent in expressiveness to Marx's [17, 18] Conditional XPath, given by adding strong until to Navigational XPath.

Filters select a subset of the nodes in an XML document, for example, the Positive XPath filters:

$$F_1 = [\text{child}::A] \quad F_2 = [\text{preceding-sibling}::A] \quad F_3 = [\text{following-sibling}::A]$$

select all nodes that have an A element as a child, left- or right-sibling.

In the context of streaming, we must consider what it means to evaluate a filter. We will consider both *root semantics* and *nodeset semantics*. In root semantics, the stream processor takes in a streamed XML document and at the close of the stream returns true or false, depending on whether or not the filter holds at the root. For example, on the stream given as:

$$S_1 = \langle B\rangle\langle C\rangle\langle/C\rangle\langle A\rangle\langle/A\rangle\langle D\rangle\langle/D\rangle\langle/B\rangle$$

the processor should return true for F_1 and false for F_2 and F_3.

In the case of a query returning a set of nodes, we will consider the *begin-tag marking problem* that produces an output stream marking the begin tags of the selected nodes. For example, the output for F_1, F_2, F_3 on input S_1 is:

$$\begin{array}{l} F_1 : \langle B^*\rangle\langle C\rangle\langle/C\rangle\langle A\rangle\langle/A\rangle\langle D\rangle\langle/D\rangle\langle/B\rangle \\ F_2 : \langle B\rangle\langle C\rangle\langle/C\rangle\langle A\rangle\langle/A\rangle\langle D^*\rangle\langle/D\rangle\langle/B\rangle \\ F_3 : \langle B\rangle\langle C^*\rangle\langle/C\rangle\langle A\rangle\langle/A\rangle\langle D\rangle\langle/D\rangle\langle/B\rangle \end{array}$$

We will also consider the corresponding *end-tag marking problem*, with output:

$$\begin{array}{l} F_1 : \langle B\rangle\langle C\rangle\langle/C\rangle\langle A\rangle\langle/A\rangle\langle D\rangle\langle/D\rangle\langle/B^*\rangle \\ F_2 : \langle B\rangle\langle C\rangle\langle/C\rangle\langle A\rangle\langle/A\rangle\langle D\rangle\langle/D^*\rangle\langle/B\rangle \\ F_3 : \langle B\rangle\langle C\rangle\langle/C^*\rangle\langle A\rangle\langle/A\rangle\langle D\rangle\langle/D\rangle\langle/B\rangle \end{array}$$

Moreover, we are interested in *zero-lookahead* algorithms for the marking problem, that generate one token of output upon reading each token of input. Note that there is no zero-lookahead algorithm for begin-tag marking of F_1 or F_3:

$$\begin{array}{l} F_1 : \langle B^?\rangle\langle C\rangle\langle/C\rangle\cdots \\ F_3 : \langle B\rangle\langle C^?\rangle\langle/C\rangle\cdots \end{array}$$

and no zero-lookahead algorithm for end-tag marking of F_3:

$$F_3 : \langle B\rangle\langle C\rangle\langle/C^?\rangle\cdots$$

We shall call filters for which zero-lookahead begin-tag or end-tag markings exist *begin-tag determined* or *end-tag determined*. From a begin-tag marking algorithm, it is trivial to produce an algorithm to output the selected nodeset in constant additional space, as no buffering is required. From an end-tag marking algorithm, the space required for buffering is proportional to the size of the largest node to be output – in many applications this will be significantly smaller than the whole input document.

There has been a significant amount of work on these problems within the database community. The most common approach has been to compile expressions into machines that use an *unbounded amount of memory* to keep track of state. They may, for example, compile an expression into a deterministic pushdown automaton (DPDA)[1]. The use of unbounded memory results from the fact

[1] For simple subsets of XPath, these DPDAs can be represented using a finite state machine [12, 1, 6]. However, a stack is still needed at runtime to store the path from the root to the current node being processed

that the set of streams that satisfy a given Navigational XPath expression, even at a fixed node, is not necessarily regular [24].

In this work we are interested in algorithms that can be done in *space and per-token time that is bounded independently of the input tree, and depending only polynomially on the expression and alphabet.* By the results above, the requirement that the space be independent of the input already requires some restriction on target trees. One key observation is that many applications that require stream-processing are concerned with content that is "data-oriented" [7]; in particular, it is common that the input data is un-nested, in the sense that an element does not occur nested inside another element with the same tag. We will restrict our attention to un-nested documents; equivalently, we assume that our trees satisfy a "non-recursive DTD" – one in which the dependency relation between tags is acyclic.

We will show that over un-nested trees $\mathcal{X}_{\text{until}}$ filters can be compiled into bounded-space machines under the root semantics, but that the bound may be exponential in the size of the formula. We will present a subset of the $\mathcal{X}_{\text{until}}$ filters that can be implemented in space usage polynomial in the formula and alphabet. We will also show that this subset is *expressively complete* for $\mathcal{X}_{\text{until}}$ over un-nested trees. We will get similar results for +HML, and for determined filters under nodeset semantics.

Our approach for getting space usage polynomial in the formula and alphabet will be to compile filters into polynomial-sized *finite state transducer networks.* This is a refinement of the approach of Olteanu [21] and Peng and Chawathe [22], where XPath expressions are compiled into a *pushdown transducer network* – consisting of pushdown automata that can output signals to other automata. A more detailed discussion of related work can be found in Section 5.

In summary, our contributions are:

- For the root semantics over un-nested trees, we identify fragments of +HML and $\mathcal{X}_{\text{until}}$ that are expressively complete, and have streaming implementations using time and space polynomial in the formula and alphabet.
- For the nodeset semantics over un-nested trees, we identify fragments of +HML and $\mathcal{X}_{\text{until}}$ that can express all begin-tag (resp. end-tag) determined queries, and have streaming begin-tag (resp. end-tag) marking implementations using both time and space polynomial in the formula and alphabet.

These results are proved for +HML and $\mathcal{X}_{\text{until}}$, but are applicable to Positive XPath and Conditional XPath.

Organization. Section 2 gives preliminaries and definitions. Sections 3 and 4 investigate streaming algorithms for boolean and nodeset queries respectively. All proofs are given in Appendix A.

2 Notation

2.1 Trees

XML documents consist of ordered labeled trees with additional data attached at nodes, either as attributes or as leaf content ('PCDATA'). In this work we will be considering filtering specifications that only deal with the ordered tree structure, so we can use a simple data model of an ordered tree:

Definition 1 (Ordered tree). *An ordered tree T with labels Σ is a finite set N together with a function $\lambda \in N \to \Sigma$ and two partial orders $\xrightarrow{\mathsf{down}^*}$, $\xrightarrow{\mathsf{right}^*} \subseteq (N \times N)$ such that:*

- $\xrightarrow{\mathsf{right}}$, $\xrightarrow{\mathsf{left}}$ *and* $\xrightarrow{\mathsf{up}}$ *are partial functions $N \to N$,*
- $\xrightarrow{\mathsf{down}} = (\xrightarrow{\mathsf{down}}\ \xrightarrow{\mathsf{right}^*}) = (\xrightarrow{\mathsf{down}}\ \xrightarrow{\mathsf{left}^*})$, *and*
- $(\xrightarrow{\mathsf{up}^*}\ \xrightarrow{\mathsf{down}^*}) = (N \times N)$,

where we write (for $\pi \in \{\mathsf{left}, \mathsf{right}, \mathsf{up}, \mathsf{down}\}$):

- $\xrightarrow{\mathsf{up}}$ *for* $\xrightarrow{\mathsf{down}^{-1}}$ *and* $\xrightarrow{\mathsf{left}}$ *for* $\xrightarrow{\mathsf{right}^{-1}}$,
- $n \xrightarrow{\pi^+} m$ *whenever* $n \xrightarrow{\pi^*} m$ *and* $n \neq m$, *and*
- $n \xrightarrow{\pi} m$ *whenever* $n \xrightarrow{\pi^+} m$ *but not* $n \xrightarrow{\pi^+} \xrightarrow{\pi^+} m$.

Note that any ordered tree has a root node n_0.

In many applications that require stream processing, the underlying documents do not have repeated instances of a tag within any downward path. This is the case, for example, of XML documents validated against a non-recursive DTD. *Most of the results of this paper will hold only for these "un-nested trees".*

Definition 2 (Un-nested tree). *An ordered tree is un-nested whenever $n \xrightarrow{\mathsf{down}^+} m$ implies $\lambda(n) \neq \lambda(m)$.*

Stream processing will deal with the standard serialization of XML documents, as a sequence of begin and end tags:

Definition 3 (Streamed tree). *Define the alphabet of a streamed tree with labels Σ as:*

$$\mathrm{Tags}(\Sigma) = \{\langle A\rangle, \langle /A\rangle \mid A \in \Sigma\}$$

For any ordered tree T with node labels Σ, define $\mathrm{stream}(T) \in (\mathrm{Tags}(\Sigma))^*$ *as* $\mathrm{stream}(n_0)$, *given by:*

$$\mathrm{stream}(n) = \langle A\rangle\, \mathrm{stream}(n_1) \ldots \mathrm{stream}(n_k)\, \langle /A\rangle$$

where $\forall i \leq k \,.\, n \xrightarrow{\mathsf{down}} n_i$ *and* $\not\leftarrow^{\mathsf{left}} n_1 \xrightarrow{\mathsf{right}} \cdots \xrightarrow{\mathsf{right}} n_k \not\xrightarrow{\mathsf{right}}$ *and* $\lambda(n) = A$.

2.2 Filtering Specifications

In this paper, we will consider specifications for nodeset queries using Marx's [17] $\mathcal{X}_{\text{until}}$ logic, which is a modal logic with a strong until operation. It extends Linear Time Temporal Logic (LTL, [8]) by allowing more than one partial order (LTL considers only one order of time). By restricting uses of until, we recover Hennessy-Milner Logic (HML) [14] as a special case.

Definition 4 ($\mathcal{X}_{\text{until}}$, HML and +HML). *Let $\mathcal{X}_{\text{until}}$ over labels Σ be defined:*

$$\phi, \psi, \chi ::= A \mid \top \mid \bot \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \pi(\phi, \psi)$$

where π ranges over $\{\mathsf{left}, \mathsf{right}, \mathsf{up}, \mathsf{down}\}$, and A ranges over Σ. The satisfaction relation for $\mathcal{X}_{\text{until}}$ is defined with the usual logical operations, together with:

- *$T, n \vDash A$ whenever $\lambda(n) = A$, and*
- *$T, n \vDash \pi(\phi, \psi)$ whenever there exists an ℓ such that $n \xrightarrow{\pi^+} \ell$ and $T, \ell \vDash \phi$ and for all m such that $n \xrightarrow{\pi^+} m \xrightarrow{\pi^+} \ell$ it holds that $T, m \vDash \psi$.*

We will write $\langle\pi\rangle\phi$ for $\pi(\phi, \bot)$ and $\langle\pi^+\rangle\phi$ for $\pi(\phi, \top)$. Let HML *be the fragment of $\mathcal{X}_{\text{until}}$ where all modalities are of the form $\langle\pi\rangle\phi$ or $\langle\pi^+\rangle\phi$. Let* +HML *be the negation-free fragment of* HML*.*

Marx [18] has shown that Conditional XPath filters (an extension of Navigational XPath with until) are equal in expressive power to $\mathcal{X}_{\text{until}}$ formulae, and that these both are equal in expressive power to first-order logic over the axis relations. Navigational XPath filters [4] are equal in expressive power to HML formulae. Positive XPath filters (negation-free Navigational XPath filters) are equal in expressive power to +HML formulae. An easy extension of Benedikt *et al.*'s argument [4] shows that +HML has the same expressive power as positive existential first-order logic over the axis relations.

We will now proceed to show results about fragments of $\mathcal{X}_{\text{until}}$, knowing that they can be applied to the appropriate fragment of Conditional XPath.

2.3 The Streaming Problem

A logical formula ϕ (in, for example, $\mathcal{X}_{\text{until}}$) defines several streaming problems.

The *root filtering problem* is to determine, given T, whether or not ϕ holds at the root. Gottlob and Koch [11] have shown that this can be done in time linear in the combined sizes of ϕ and T, if one allows the tree T to be stored in memory. In contrast, we want an algorithm that has limited access to T.

A *root stream processor* is a Turing machine *TM* with one input tape and one working tape, such that *TM* can only move forward on its input tape. Such a *TM* is a *root streaming implementation* of ϕ if *TM* accepts on input stream(T) iff T satisfies ϕ at the root. The *runtime space usage* of such a *TM* on an input s is the number of workspace tape elements used. The *total space usage* is the runtime space usage plus the size of the *TM*. The *per-token time usage* of such a *TM* on an input s is the number of steps taken, divided by $|s|$.

In Section 3, we will show that every formula has a root streaming implementation with total space and per-token time that is independent of the tree. Implementations which use polynomial total space and per-token time do not exist for every formula, but we will find a fragment of $\mathcal{X}_{\text{until}}$ which does support polynomial implementation, and moreover with no loss of expressive power.

We now turn to nodeset queries given by filters – that is to filters not restricted to the root node. In main-memory processing, the entire set of subtrees of nodes satisfying the filter would be returned. In a streaming setting, we may be interested in an output stream that includes indicators of which nodes are in the solution nodeset. We will consider adding these indicators to either the begin tags or to the end tags.

Definition 5 (Streamed document tree with selected begin tags). *For any ordered tree T with node labels Σ, and any formula ϕ, define* $\text{bstream}(T, \phi) \in (\text{Tags}(\Sigma) \times 2)^*$ *as* $\text{bstream}(n_0, \phi)$*, given by:*

$$\text{bstream}(n, \phi) = (\langle A \rangle, b)\, \text{bstream}(n_1, \phi) \ldots \text{bstream}(n_k, \phi)\, (\langle /A \rangle, \bot)$$

where $\forall i \leq k \,.\, n \xrightarrow{\text{down}} n_i$ *and* $\not\leftarrow n_1 \xrightarrow{\text{right}} \cdots \xrightarrow{\text{right}} n_k \not\xrightarrow{\text{right}}$ *and* $\lambda(n) = A$ *and* $T, n \vDash \phi \leftrightarrow b$ *(where* $2 = \{\top, \bot\}$*, the boolean constants)*

The *begin-tag filtering problem* is, given as input ϕ and $\text{stream}(T)$, to output $\text{bstream}(T, \phi)$. We can similarly define the *end-tag filtering problem*, defining the stream $\text{estream}(T, \phi)$ analogously to bstream above, but with booleans annotating end-tags.

A *nodeset stream processor* is a Turing machine *TM* with one read-only input tape, one working tape, and one write-only output tape such that *TM* can only move forward on its input tape, and only add symbols to the end of its output tape. Such a processor has *zero-lookahead* if it produces exactly one output symbol whenever it moves its head on the input tape. Such a processor *TM* is a *begin-tag streaming implementation* of ϕ if *TM* outputs $\text{bstream}(T, \phi)$. We can similarly talk about an *end-tag streaming implementation*. The notions of space and per-token time efficiency in a processor are as before.

In Section 4, we will show that not every formula has a begin-tag or end-tag streaming implementation with total space and per-token time that is independent of the tree. Again, we will find a fragment of $\mathcal{X}_{\text{until}}$ which does admit efficient implementations, with no loss of expressive power.

3 Filtering of Boolean Queries

We first show that every formula has a root streaming implementation with total space and per-token time independent of the input tree.

Proposition 1. *For every $\mathcal{X}_{\text{until}}$ formula ϕ over labels Σ there is a number k and a root streaming implementation $TM_{\phi,\Sigma}$ over un-nested ordered trees with labels Σ using at most k total space and per-token time.*

Even for simple queries, we may not be able to get space-efficient implementations. Consider the formulae ϕ_n over labels $\{A, B, C, T_1, F_1, \ldots, T_n, F_n\}$ defined:

$$\begin{aligned}\phi_n &= \langle \mathsf{down} \rangle (A \wedge \psi_1 \wedge \cdots \wedge \psi_n) \\ \psi_i &= (\langle \mathsf{down} \rangle T_i \wedge \langle \mathsf{right}^+ \rangle (B \wedge \langle \mathsf{down} \rangle T_i)) \\ &\quad \vee (\langle \mathsf{down} \rangle F_i \wedge \langle \mathsf{right}^+ \rangle (B \wedge \langle \mathsf{down} \rangle F_i))\end{aligned}$$

evaluated over trees with streaming representations of the form:

$$\begin{aligned}&\langle C \rangle \langle A \rangle s_1 \langle / A \rangle \cdots \langle A \rangle s_k \langle / A \rangle \langle B \rangle s \langle / B \rangle \langle / C \rangle \\ &\quad \text{where } s, s_1, \ldots, s_k \in \{\langle T_1 / \rangle, \langle F_1 / \rangle\} \times \cdots \times \{\langle T_n / \rangle, \langle F_n / \rangle\}\end{aligned}$$

It is clear that such a tree satisfies ϕ_n precisely when $s \in \{s_1, \ldots, s_k\}$, and there are 2^{2^n} such sets, and so there is no polynomial space implementation of +HML:

Proposition 2. *There is no subexponential F such that every* +HML *formula ϕ over labels Σ has a root streaming implementation $TM_{\phi,\Sigma}$ over un-nested ordered trees with labels Σ using at most $F(|\phi|, |\Sigma|)$ total space.*

We must thus look for a sublanguage of $\mathcal{X}_{\text{until}}$ that has efficient implementations.

The notion of a subformula of a formula is as usual. A top-level subformula is one which does not occur inside a subformula of the form $\pi(\phi, \psi)$.

Definition 6 (Backward $\mathcal{X}_{\text{until}}$). *Backward $\mathcal{X}_{\text{until}}$ is the fragment of $\mathcal{X}_{\text{until}}$ in which:*

- *all occurrences of* up *are of the form* $\mathsf{up}(\phi, \psi)$*, where ϕ and ψ have no top-level occurrences of* down*, and*
- *all occurrences of* right *are of the form* $\mathsf{down}(\phi \wedge \neg\langle \mathsf{right} \rangle \top, \psi)$.

Note that the restriction on right disallows examples such as those used in the proof of Proposition 2, and that the restriction on up bans the similar formula where $\langle \mathsf{right}^+ \rangle$ is replaced by $\langle \mathsf{up} \rangle \langle \mathsf{down} \rangle$. Also note that we cannot completely ban right, as there is no right-free backward equivalent of $\langle \mathsf{down} \rangle (A \wedge \neg\langle \mathsf{right} \rangle \top)$ ("my last child is an A"). Our first main result is:

Theorem 1. *There is a polynomial P such that every backward $\mathcal{X}_{\text{until}}$ formula ϕ over labels Σ has a root streaming implementation $TM_{\phi,\Sigma}$ over un-nested ordered trees with labels Σ using at most $P(|\phi|, |\Sigma|)$ total space and per-token time. Furthermore, one can produce $TM_{\phi,\Sigma}$ from ϕ and Σ in polynomial time.*

The construction of $TM_{\phi,\Sigma}$ is given by building an appropriate *synchronous transducer network*, an acyclic collection of synchronous transducers [5] where the output of one transducer is allowed as input to another. Transducers whose input-output relation is a function are called *sequential*, and networks built from sequential transducers generate deterministic automata, so can be executed in polynomial space and per-token time.

The construction makes use of *named* $\mathcal{X}_{\text{until}}$ formulae, which require every modality to specify the label of one of the nodes involved (for down and up, the parent node is named, and for left and right, the parent of the context node is).

Definition 7 (Named $\mathcal{X}_{\text{until}}$). *Named $\mathcal{X}_{\text{until}}$ is the fragment of $\mathcal{X}_{\text{until}}$ in which:*

- *all occurrences of* down *are of the form* $A \wedge \mathsf{down}(\phi, \psi)$*, where* $A \in \Sigma$
- *all occurrences of* up *are of the form* $\mathsf{up}(A \wedge \phi, \psi)$*,*
- *all occurrences of* left *are of the form* $\langle \mathsf{up} \rangle A \wedge \mathsf{left}(\phi, \psi)$*, and*
- *all occurrences of* right *are of the form* $\langle \mathsf{up} \rangle A \wedge \mathsf{right}(\phi, \psi)$*.*

Theorem 2. *Every $\mathcal{X}_{\text{until}}$ formula ϕ over labels Σ has an implementation $TN_{\phi,\Sigma}$ as a network of $O(|\phi| \times |\Sigma|)$ synchronous transducers, each of which has $O(|\Sigma|)$ states. If ϕ is in named $\mathcal{X}_{\text{until}}$, then $TN_{\phi,\Sigma}$ contains $O(|\phi|)$ transducers. If ϕ is in backward $\mathcal{X}_{\text{until}}$, then $TN_{\phi,\Sigma}$ is sequential. Furthermore, $TN_{\phi,\Sigma}$ can be constructed in polynomial time.*

What do we give up from staying within backward $\mathcal{X}_{\text{until}}$? The next result shows that, in terms of expressiveness over un-nested trees, we lose nothing, and in fact we can be even more restrictive, and only require downward formulae:

Definition 8 (Downward $\mathcal{X}_{\text{until}}$). *Downward $\mathcal{X}_{\text{until}}$ is the fragment of $\mathcal{X}_{\text{until}}$ in which:*

- *there are no occurrences of* up*, and*
- *there are no top-level occurrences of* left *or* right*.*

Theorem 3. *Every $\mathcal{X}_{\text{until}}$ formula ϕ has a backward downward $\mathcal{X}_{\text{until}}$ formula ψ which agrees with ϕ on the root node of any un-nested ordered tree.*

The proof makes use of an analog of Marx's variant [17] of Gabbay's Separation Theorem [9] for ordered trees, showing that $\mathcal{X}_{\text{until}}$ formulas can be rewritten into "strict backward", "strict forward", and "backward downward" formulae. For formulae evaluated at the root node, we can then eliminate the strict backward and forward components. A similar completeness result holds within positive HML, but without any restriction on nesting:

Theorem 4. *For every* +HML *formula ϕ there is a backward downward* +HML *formula ψ which agrees with ϕ on the root node of any ordered tree.*

This result is proven using a simpler argument, a variant of that used in [20] and Theorem 5.1 of [4]. We translate +HML queries to logical formulas, and then show that these formulas can be normalized to be of a special form. This normal form is a variant of the "tree pattern queries" of [4]. Given a normalized formula, we can apply root-equivalence, end-equivalence, or begin-equivalence to the normalized formula, arriving at a logical formula in which all the bound variables are restricted to lie in a certain relation to the free variable. Finally, we translate the syntactic restrictions back into +HML, where they produce a formula that is backward and downward. It is interesting that the analogous completeness result does not hold for HML (or for Navigational XPath).

Theorem 5. *The* HML *filter $\langle \mathsf{down} \rangle (B \wedge \neg \langle \mathsf{right}^+ \rangle A)$ is not equivalent to any filter in backward* HML*.*

The proof uses trees T and T' parameterized by a bound K:

$$\text{stream}(T) = \langle R\rangle(\langle A/\rangle\langle C/\rangle^{K-1}\langle B/\rangle\langle C/\rangle^{K-1})^K\langle A/\rangle\langle C/\rangle^{K-1}\langle /R\rangle$$
$$\text{stream}(T') = \langle R\rangle(\langle A/\rangle\langle C/\rangle^{K-1}\langle B/\rangle\langle C/\rangle^{K-1})^K\langle /R\rangle$$

Clearly the formula $\langle \mathsf{down}\rangle(B \wedge \neg\langle \mathsf{right}^+\rangle A)$ is false at the root of T and true at the root of T'. Using a bisimulation argument, we can show that no backward formula with size bounded by K can distinguish T from T'.

4 Filtering of Nodeset Queries

We now turn to nodeset queries, and begin with a negative result. Even without requiring zero-lookahead, it is not always possible to implement filters (for example $\langle \mathsf{right}^+\rangle A$) in space independent of the tree.

Proposition 3. *There is a* +HML *formula ϕ over labels Σ such that for no k is there a begin-tag or end-tag streaming implementation TM that uses at most k total space over un-nested ordered trees with labels Σ.*

We shall call the formulae which have zero-lookahead end-tag streaming implementations "end-tag determined", and similarly for "begin-tag determined".

Definition 9 (Determined formulae). *For any tree T with node $n \in T$, the subtrees* $\text{btree}(T, n)$ *and* $\text{etree}(T, n)$ *are such that:*

$$n' \in \text{btree}(T, n) \text{ whenever } n \xrightarrow{\mathsf{up}^*} n' \text{ or } n \xrightarrow{\mathsf{up}^*} \xrightarrow{\mathsf{left}^+} \xrightarrow{\mathsf{down}^*} n' \text{ in } T$$
$$n' \in \text{etree}(T, n) \text{ whenever } n' \in \text{btree}(T, n) \text{ or } n \xrightarrow{\mathsf{down}^*} n' \text{ in } T$$

A formula ϕ is end-tag determined whenever, for all $n \in T$ and $n' \in T'$ with $\text{etree}(T, n)$ *isomorphic to* $\text{etree}(T', n')$*, we have $T, n \vDash \phi$ precisely when $T', n' \vDash \phi$. The begin-tag determined formulae are defined similarly.*

It is easy to see that a filter has a zero-lookahead end-tag (resp. begin-tag) streaming implementation precisely when it is end-tag (resp. begin-tag) determined. It is also easy to see that backward $\mathcal{X}_{\text{until}}$ formulae are end-tag determined, since they only look at the nodes in the end-tag preceding subtree of the input node, and that *strict backward* $\mathcal{X}_{\text{until}}$ formulae are begin-tag determined:

Definition 10 (Strict backward $\mathcal{X}_{\text{until}}$). *A formula is in strict backward $\mathcal{X}_{\text{until}}$ if it is in backward $\mathcal{X}_{\text{until}}$ and has no top-level occurrences of* down*.*

Our transducer network results show that backward (resp. strict backward) $\mathcal{X}_{\text{until}}$ formulae have efficient end-tag (resp. begin-tag) streaming implementations:

Theorem 6. *There is a polynomial P such that every backward (resp. strict backward) $\mathcal{X}_{\text{until}}$ formula ϕ over labels Σ has an end-tag (resp. begin-tag) streaming implementation $TM_{\phi,\Sigma}$ over un-nested ordered trees with labels Σ using at most $P(|\phi|, |\Sigma|)$ total space and per-token time. Furthermore, one can produce $TM_{\phi,\Sigma}$ from ϕ and Σ in polynomial time.*

The notion of begin-tagged and end-tagged determined turns out to be decidable: convert a formula into a deterministic automaton with no sink states, then check whether any state has transitions on both marked and unmarked variants of the same tag. However, checking that a formula is determined cannot be done efficiently; it can be shown, by reduction to the satisfiability problem for XPath [3], that the problem is PSPACE-hard. We now show that working within backward $\mathcal{X}_{\text{until}}$ does not restrict our ability to express determined queries, and in fact we can be even more restrictive, requiring only *oscillation-free* formulae:

Definition 11 (Oscillation-free $\mathcal{X}_{\text{until}}$). *Oscillation-free $\mathcal{X}_{\text{until}}$ is the fragment of $\mathcal{X}_{\text{until}}$ in which all occurrences of* down *contain no occurrences of* up.

Theorem 7. *Every end-tag (resp. begin-tag) determined $\mathcal{X}_{\text{until}}$ formula ϕ has a backward (resp. strict backward) oscillation-free $\mathcal{X}_{\text{until}}$ formula ψ which agrees with ϕ on any node of any un-nested ordered tree.*

The proof is similar to that of Theorem 3. For positive HML, we can again get a stronger completeness result:

Theorem 8. *Every end-tag (resp. begin-tag) determined* +HML *formula ϕ has an backward (resp. strict backward) oscillation-free* +HML *formula ψ which agrees with ϕ on any node of any ordered tree.*

This result also uses a rewriting argument, analogous to those of Benedikt *et al.* [4] or Olteanu [20]. The analogous completeness results do not hold for HML (for example, it is not true that end-tag determined HML formulas can be rewritten into backward HML) – the argument is along the lines of Theorem 5.

5 Related work

Much of the preceding work deals with XPath *expressions* rather than filters; expressions are functions that take a node and return a nodeset: for example descendant::A returns all A-tagged descendants of a given node. It is known from Marx [19] that the expressiveness of Navigational XPath filters is the same as that of Navigational XPath expressions evaluated at the root node. This distinction between filters and expressions is what accounts for the emphasis on reverse axes in our work, versus forward axes in the work of Olteanu [20].

As mentioned above, work on XPath filtering generally assumes that documents may have nested tags, and thus looks for streaming models that require an unbounded stack. Bar-Yossef *et al.* [2] and Grohe *et al.* [13] prove lower-bounds on the memory usage in streaming algorithms; for example Grohe *et al.* show that any streaming algorithm for XPath on general XML documents requires space at least proportional to the tree depth.

In contrast, there has been work on constant-space evaluation of constraints expressed by DTDs and XML Schemas. Segoufin and Vianu [24] investigate which DTDs can be validated in constant space on streams, and observes that a DTD can be validated in constant space if all trees that satisfy it are un-nested.

Begin-tag and end-tag determined XPath filters have not previously been investigated, although they have been studied in the context of XML Schemas by Martens *et al.* [16] and Madhusadan *et al.* [15].

The two main components of our work: transducer networks and rewriting, both appear in the work of Olteanu. His use of rewriting [20] is to eliminate reverse axes within an XPath-like language over general trees. Our Theorem 4 is thus a variant of his result, and in Theorem 3 we show that this phenomena extends to the much richer language $\mathcal{X}_{\text{until}}$, provided that we restrict to un-nested trees. Our Theorem 5 shows that this elimination cannot be done within full Navigational XPath, even over un-nested trees. Although this appears to contradict Corollary 5.2 of [20], the term "XPath" in that corollary is used to refer to a language LGQ, which is closer in expressiveness to Positive XPath rather than XPath. Our use of transducer networks extends Olteanu's work in [21], which works over general XML documents, and hence the networks are DPDAs rather than DFAs. The networks are used for the forward fragment of positive XPath. Our results show that the construction extends to the much more expressive language $\mathcal{X}_{\text{until}}$, and that it provides a finite state transducer network when restricted to un-nested trees.

Our rewriting of $\mathcal{X}_{\text{until}}$ filters makes use of a separation result very similar to Theorem 8 of Marx [17]. Marx's result is over general trees, and does not separate filters that look "to the left and up" from those that look "to the right and up" – such a separation is needed for our result on un-nested trees, but does not hold in general. Unfortunately, an error has been found in the proof of Theorem 8 in [17] – Lemma 10 of that paper includes a distributivity property $(\mathsf{down}(\phi, \psi \wedge \chi) = \mathsf{down}(\phi, \psi) \wedge \mathsf{down}(\phi, \chi))$ which is only true when $\xrightarrow{\mathsf{down}}$ is deterministic. As a result, his induction (in an un-numbered "final step" at the end of Section 4) fails. Semantic separation has been shown, using Marx's expressive completeness result for Conditional XPath [18]. But this proof does not imply syntactic separation for $\mathcal{X}_{\text{until}}$.

Our completeness results for boolean queries can be seen as extensions to the ordered tree setting of the well-known fact that LTL with only future operators has the same expressiveness as LTL with both past and future, if one considers only the initial node of a string. Transducer networks have been utilized several times in the verification literature (e.g. Pnueli and Zaks [23]), but their use in conjunction with reverse-direction fragments is, to our knowledge, new.

References

1. M. Altinel and M. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proc. 26th International Conference on Very Large Data Bases (VLDB)*, pages 53–64, 2000.
2. Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 177–188, 2004.

3. M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *Proc. 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, 2005.
4. M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. *Theoretical Computer Science*, 336(1):3–31, 2005.
5. J. Besterel and D. Perrin. Algorithms on words. In M. Lothaire, editor, *Applied Combinatorics on Words*, chapter 1. Cambridge University Press, 2005.
6. C. Y. Chan, P. Felber, M. N. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. In *Proc. 18th IEEE International Conference on Data Engineering (ICDE)*, 2002.
7. B. Choi. What are real DTDs like. In *Proc. Fifth International Workshop on the Web and Databases (WebDB)*, 2002.
8. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.
9. D. Gabbay. Expressive functional completeness in tense logic. In U. Mönnich, editor, *Aspects of Philosophical Logic*, pages 67–89, 1981.
10. D. Gabbay. Declarative past and imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proc. Colloquium on Temporal Logic in Specification*, pages 67–89, 1989.
11. G. Gottlob and C. Koch. Monadic datalog and the expressive power of web information extraction languages. *Journal of the ACM*, 51(1):74–113, 2004.
12. T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata. In *Proc. 9th International Conference on Database Theory (ICDT)*, 2003.
13. M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. In *Proc. International Conference on Automata, Languages and Programming (ICALP)*, 2005.
14. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.
15. V. Kumar, P. Madhusadan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *WWW*, 2007.
16. W. Martens, F. Neven, and T. Schwentick. Which XML schemas admit 1-pass pre-order traversal. In *Proc. 10th International Conference on Database Theory (ICDT)*, 2005.
17. M. Marx. Conditional XPath, the first order complete XPath dialect. In *Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 13–22, 2004.
18. M. Marx. Conditional XPath. *ACM Transactions on Database Systems*, pages 929–959, 2005.
19. M. Marx. First order paths in ordered trees. In *Proc. 10th International Conference on Database Theory (ICDT)*, 2005.
20. D. Olteanu. Forward node-selecting queries over trees. *ACM TODS*, 32(1), 2007.
21. D. Olteanu. Streamed and progressive evaluation of XPath. *IEEE Transactions on Knowledge and Data Engineering*, 19(7), July 2007.
22. F. Peng and S. Chawathe. XPath queries on streaming data. In *Proc. 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2003.
23. A. Pnueli and A. Zaks. PSL model checking and runtime verification via testers. In *Proc. International Symposium on Formal Methods (FM)*, 2006.
24. L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, 2002.

25. World Wide Web Consortium. XML path language (XPath) 2.0: W3C recommendation, 2007. http://www.w3.org/TR/xpath20/.

A Detailed Proofs

A.1 Transducer Networks and Proofs of Efficiency

In this appendix, we give the formal definitions of transducer networks, and show two results:

- Theorem 9, which shows how a transducer network can be built from an $\mathcal{X}_{\text{until}}$ formula, and
- Proposition 5, which sketches how a streaming implementation can be built from a transducer network.

From these we get:

- Theorem 2, which follows directly from Theorem 9,
- Theorem 6, which follows directly from Theorem 2 and Proposition 5,
- Theorem 1 which follows from Theorem 6 by taking the end-tag streaming implementation, ignoring the output tape, and accepting whenever the last output end mark is $\top$, and
- Proposition 1, which follows from Theorem 9 by building a (possibly non-sequential) transducer network, flattening it to a (possibly nondeterministic) automaton, determinizing the automaton, and considering the automaton as a Turing Machine. This construction has double-exponential blowup, but has total space and per-token time independent of the input tree.

We begin with the formal definitions underlying the transducer networks that witness our efficient implementations.

Definition 12 (Automaton). *An automaton $\mathcal{A}$ over alphabet Σ is a quadruple $(Q, \longrightarrow, I, F)$ where Σ and Q are finite sets (the alphabet and state set respectively), $\longrightarrow \subseteq Q \times \Sigma \times Q$ (the transition relation), and $I, F \subseteq Q$ (the initial and final states respectively). Write $q \xrightarrow{\ell} q'$ for $(q, \ell, q') \in \longrightarrow$. For $s \in \Sigma^*$, write $q \stackrel{s}{\Longrightarrow} q'$ for the transitive reflexive closure of $\longrightarrow$, that is:*

$$q \stackrel{\ell_1 \ldots \ell_n}{\Longrightarrow} q' \text{ whenever } q = q_0 \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_n} q_n = q'$$

The language induced by an automaton $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{s \mid I \ni q \stackrel{s}{\Longrightarrow} q' \in F\}$. An automaton is deterministic whenever the transition relation is a partial function $\longrightarrow : Q \times \Sigma \to Q$ and the first-move relation $\text{first} = \{(\ell, q') \mid I \ni q \xrightarrow{\ell} q'\}$ is a partial function $\text{first} : \Sigma \to Q$.

Definition 13 (Synchronous transducer). *A synchronous transducer $\mathcal{T}$ over input alphabet Σ and output alphabet Δ is an automaton over alphabet $\Sigma \times \Delta$.*

Write $q \xrightarrow{\ell/k} q'$ *for* $(q,(\ell,k),q') \in \longrightarrow$. *For* $s \in \Sigma^*$ *and* $t \in \Delta^*$, *write* $q \stackrel{s/t}{\Longrightarrow} q'$ *for the transitive reflexive closure of* $\longrightarrow$, *that is:*

$$q \xrightarrow{\ell_1 \ldots \ell_n / k_1 \ldots k_n} q' \text{ whenever } q = q_0 \xrightarrow{\ell_1/k_1} \cdots \xrightarrow{\ell_n/k_n} q_n = q'$$

The relation induced by a transducer $\mathcal{T}$ *is* $\mathcal{R}(\mathcal{T}) = \{(s,t) \mid I \ni q \stackrel{s/t}{\Longrightarrow} q' \in F\}$. *A transducer is sequential whenever the transition relation is a partial function* $\longrightarrow : Q \times \Sigma \to \Delta \times Q$ *and the first-move relation is a partial function* first : $\Sigma \to \Delta \times Q$.

Proposition 4. *For any synchronous transducers* $\mathcal{T}$ *and* $\mathcal{T}'$ *we can construct the synchronous transducers:*

- id *such that* $\mathcal{R}(\mathrm{id}) = \{(s,s) \mid s \in \Sigma^*\}$,
- $\mathcal{T};\mathcal{T}'$ *such that* $\mathcal{R}(\mathcal{T};\mathcal{T}') = \{(s,t) \mid (s,u) \in \mathcal{R}(\mathcal{T}), (u,t) \in \mathcal{R}(\mathcal{T}')\}$, *and*
- $\langle \mathcal{T}, \mathcal{T}' \rangle$ *such that* $\mathcal{R}(\langle \mathcal{T}, \mathcal{T}' \rangle) = \{(s,t) \mid (s, \pi_1^*(t)) \in \mathcal{R}(\mathcal{T}), (s, \pi_2^*(t)) \in \mathcal{R}(\mathcal{T}')\}$.

If $\mathcal{T}$ *and* $\mathcal{T}'$ *are sequential, then so are* id, $\mathcal{T};\mathcal{T}'$ *and* $\langle \mathcal{T}, \mathcal{T}' \rangle$.

Definition 14 (Synchronous transducer network). *A synchronous transducer network with generators* $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ *is a synchronous transducer definable by:*

$$\mathcal{N} ::= \mathcal{T}_i \mid \mathcal{N};\mathcal{N} \mid \langle \mathcal{N}, \mathcal{N} \rangle$$

We shall write $|\mathcal{N}|$ *for the number of transducers in* $\mathcal{N}$, *given by* $|\mathcal{T}_i| = 1$ *and* $|\mathcal{N};\mathcal{N}'| = |\langle \mathcal{N}, \mathcal{N}' \rangle| = |\mathcal{N}| + |\mathcal{N}'|$.

The significance of synchronous transducers is that they provide efficient streaming implementations:

Proposition 5. *Let* $\mathcal{N}$ *be a transducer network consisting of only sequential transducers. Let* q *be the total number of states of any transducer in the network, and let* s *the total number of input or output symbols of any transducer in the network. Then the partial function induced by* $\mathcal{N}$ *can be implemented as a zero-lookahead nodeset stream processor with total space and per-token time at most polynomial in* q, s *and* $|\mathcal{N}|$.

Proof (sketch). We build a machine that simply runs the transducer network, storing the vector of component states on the worktape. The runtime space used is only the state of each component in the transducer network, and the representation of the machine just needs to record the transition function of each component. To get zero-lookahead, we precompute the output symbol for each possible input symbol, so the output can take place immediately after the input. □

We now show the core of the efficiency results, which is that $\mathcal{X}_{\text{until}}$ formulae can be implemented as polynomial-sized transducer networks:

Theorem 9. *For any $\mathcal{X}_{\text{until}}$ formula ϕ over Σ, we can construct synchronous transducer networks $\mathcal{B}[\![\phi]\!]$ and $\mathcal{E}[\![\phi]\!]$ such that:*

- $\langle \mathrm{id}, \mathcal{B}[\![\phi]\!] \rangle$ *induces a function which maps* $\mathrm{stream}(T)$ *to* $\mathrm{bstream}(T, \phi)$*, and*
- $\langle \mathrm{id}, \mathcal{E}[\![\phi]\!] \rangle$ *induces a function which maps* $\mathrm{stream}(T)$ *to* $\mathrm{estream}(T, \phi)$*.*

Moreover:

- $\mathcal{B}[\![\phi]\!]$ *and* $\mathcal{E}[\![\phi]\!]$ *contain* $O(|\phi| \times |\Sigma|)$ *transducers, each with* $O(|\Sigma|)$ *states, input symbols and output symbols,*
- *if* ϕ *is in named* $\mathcal{X}_{\text{until}}$ *then* $\mathcal{B}[\![\phi]\!]$ *and* $\mathcal{E}[\![\phi]\!]$ *contain* $O(|\phi|)$ *transducers,*
- *if* ϕ *is in backward* $\mathcal{X}_{\text{until}}$ *and has no top-level uses of* down *then* $\mathcal{B}[\![\phi]\!]$ *is sequential, and*
- *if* ϕ *is in backward* $\mathcal{X}_{\text{until}}$ *then* $\mathcal{E}[\![\phi]\!]$ *is sequential.*

Proof. In giving descriptions of transducers, we will elide many no-op transitions. Formally, we define the *completion* of a synchronous transducer $\mathcal{T}$ over input alphabet Σ and output alphabet 2 by adding transitions:

$$q \xrightarrow{\ell/\bot} q \text{ whenever } \nexists c, q' \,.\, q \xrightarrow{\ell/c} q' \text{ in } \mathcal{T} \qquad (\ell \in \Sigma)$$

We now show how to construct $\mathcal{B}[\![\phi]\!]$ in the case of named strict backward $\mathcal{X}_{\text{until}}$:

- $\mathcal{B}[\![\phi \vee \psi]\!]$ is defined to be $\langle \mathcal{B}[\![\phi]\!], \mathcal{B}[\![\psi]\!] \rangle ; \mathcal{B}[\![\vee]\!]$ where $\mathcal{B}[\![\vee]\!]$ the transducer with transitions:
$$0 \xrightarrow{\top,b/\top} \quad 0 \xrightarrow{b,\top/\top} \quad 0 \xrightarrow{\bot,\bot/\bot}$$
where 0 is the initial and accepting state. We will write $\mathcal{T}_1 \vee \mathcal{T}_2$ as shorthand for $\langle \mathcal{T}_1, \mathcal{T}_2 \rangle ; \mathcal{B}[\![\vee]\!]$. The other boolean operations are defined similarly.
- $\mathcal{B}[\![A]\!]$ is given by completing the transducer with transitions:
$$0 \xrightarrow{\langle A \rangle/\top} 0$$
where 0 is the initial and accepting state. $\mathcal{E}[\![A]\!]$ is defined similarly.
- $\mathcal{B}[\![\langle \mathsf{up} \rangle A \wedge \mathsf{left}(\phi, \psi)]\!]$ is defined to be $\langle \mathrm{id}, \langle \mathcal{E}[\![\phi]\!], \mathcal{E}[\![\psi]\!] \rangle \rangle ; \mathcal{B}[\![\mathsf{left}, A]\!]$ where $\mathcal{B}[\![\mathsf{left}, A]\!]$ is given by completing the transducer with transitions (for every $B \neq A$):
$$\begin{array}{llll}
0 \xrightarrow{\langle A \rangle,\bot,\bot/\bot} 1 & & & \\
1 \xrightarrow{\langle /A \rangle,b,c/\bot} 0 & 1 \xrightarrow{\langle B \rangle,\bot,\bot/\bot} B_1 & & \\
B_1 \xrightarrow{\langle /B \rangle,\top,b/\bot} 2 & B_1 \xrightarrow{\langle /B \rangle,\bot,b/\bot} 1 & & \\
2 \xrightarrow{\langle /A \rangle,b,c/\bot} 0 & 2 \xrightarrow{\langle B \rangle,\bot,\bot/\top} B_2 & & \\
B_2 \xrightarrow{\langle /B \rangle,\bot,\bot/\bot} 1 & B_2 \xrightarrow{\langle /B \rangle,\top,b/\bot} 2 & B_2 \xrightarrow{\langle /B \rangle,b,\top/\bot} 2 &
\end{array}$$
where 0 is the initial and accepting state.
- $\mathcal{B}[\![\mathsf{up}(A \wedge \phi, \psi)]\!]$ is defined to be $\langle \mathrm{id}, \langle \mathcal{B}[\![\phi]\!], \mathcal{B}[\![\psi]\!] \rangle \rangle ; \mathcal{B}[\![\mathsf{up}, A]\!]$ where $\mathcal{B}[\![\mathsf{up}, A]\!]$ is given by completing the transducer with transitions (for every $B \neq A$):
$$\begin{array}{lll}
0 \xrightarrow{\langle A \rangle,\top,b/\bot} 1 & & \\
1 \xrightarrow{\langle /A \rangle,\bot,\bot/\bot} 0 & 1 \xrightarrow{\langle B \rangle,b,\top/\top} 1 & 1 \xrightarrow{\langle B \rangle,b,\bot/\top} B \\
B \xrightarrow{\langle /B \rangle,\bot,\bot/\bot} 1 & &
\end{array}$$
where 0 is the initial and accepting state.

We simultaneously define $\mathcal{E}[\![\phi]\!]$ in the case of named backward $\mathcal{X}_{\text{until}}$; most cases are similar to $\mathcal{B}[\![\phi]\!]$, but we now provide definitions for formulae with top-level occurrences of down:

- $\mathcal{E}[\![A \wedge \mathsf{down}(\phi, \psi)]\!]$ is defined to be $\langle \mathrm{id}, \langle \mathcal{E}[\![\phi]\!], \mathcal{E}[\![\psi]\!] \rangle \rangle; \mathcal{E}[\![\mathsf{down}, A]\!]$ where $\mathcal{E}[\![\mathsf{down}, A]\!]$ is given by completing the transducer with transitions (for every $B \neq A$):

$$\begin{array}{llll} 0 \xrightarrow{\langle A\rangle,\bot,\bot/\bot} 1 & & & \\ 1 \xrightarrow{\langle/A\rangle,b,c/\bot} 0 & 1 \xrightarrow{\langle/B\rangle,\top,b/\bot} 2 & & \\ 2 \xrightarrow{\langle/A\rangle,b,c/\top} 0 & 2 \xrightarrow{\langle/B\rangle,\bot,\bot/\bot} 1 & 2 \xrightarrow{\langle B\rangle,\bot,\bot/\bot} B & \\ B \xrightarrow{\langle/B\rangle,b,c/\bot} 2 & & & \end{array}$$

  where 0 is the initial and accepting state.
- $\mathcal{E}[\![A \wedge \mathsf{down}(\phi \wedge \neg\langle \mathsf{right}\rangle \top, \psi)]\!]$ is defined to be $\langle \mathrm{id}, \langle \mathcal{E}[\![\phi]\!], \mathcal{E}[\![\psi]\!] \rangle \rangle; \mathcal{E}[\![\mathrm{downlast}, A]\!]$ where $\mathcal{E}[\![\mathrm{downlast}, A]\!]$ is given by completing the transducer with transitions (for every $B \neq A$):

$$\begin{array}{llll} 0 \xrightarrow{\langle A\rangle,\bot,\bot/\bot} 1 & & & \\ 1 \xrightarrow{\langle/A\rangle,b,c/\bot} 0 & 1 \xrightarrow{\langle/B\rangle,\top,b/\bot} 3 & & \\ 2 \xrightarrow{\langle/A\rangle,b,c/\top} 0 & 2 \xrightarrow{\langle/B\rangle,\bot,\bot/\bot} 1 & 2 \xrightarrow{\langle B\rangle,\bot,\bot/\bot} B & \\ 3 \xrightarrow{\langle B\rangle,\bot,\bot/\bot} 1 & 3 \xrightarrow{\langle/B\rangle,\bot,\bot/\bot} 1 & 3 \xrightarrow{\langle/B\rangle,b,\top/\bot} 2 & \\ B \xrightarrow{\langle/B\rangle,b,c/\bot} 2 & & & \end{array}$$

  where 0 is the initial and accepting state.

For named formulae that are not backward, we extend the definitions as follows:

- $\mathcal{E}[\![\langle \mathsf{up}\rangle A \wedge \mathsf{right}(\phi, \psi)]\!]$ is defined to be $\langle \mathrm{id}, \langle \mathcal{B}[\![\phi]\!], \mathcal{B}[\![\psi]\!] \rangle \rangle; \mathcal{E}[\![\mathsf{right}, A]\!]$ where $\mathcal{E}[\![\mathsf{right}, A]\!]$ is the transducer with transitions:

$$\begin{array}{l} q \xrightarrow{\langle A\rangle,b,c/d} q' \text{ in } \mathcal{E}[\![\mathsf{right}, A]\!] \text{ whenever } q \xleftarrow{\langle/A\rangle,b,c/d} q' \text{ in } \mathcal{E}[\![\mathsf{left}, A]\!] \\ q \xrightarrow{\langle/A\rangle,b,c/d} q' \text{ in } \mathcal{E}[\![\mathsf{right}, A]\!] \text{ whenever } q \xleftarrow{\langle A\rangle,b,c/d} q' \text{ in } \mathcal{E}[\![\mathsf{left}, A]\!] \end{array}$$

  and the same initial and acceptor state as $\mathcal{E}[\![\mathsf{left}, A]\!]$. $\mathcal{B}[\![\langle \mathsf{up}\rangle A \wedge \mathsf{right}(\phi, \psi)]\!]$ is defined similarly. Note that this transducer is not sequential.
- $\mathcal{B}[\![A \wedge \mathsf{down}(\phi, \psi)]\!]$ is defined to be $\mathcal{E}[\![A \wedge \mathsf{down}(\phi, \psi)]\!]; \Omega(A)$ where $\Omega(A)$ is given by completing the transducer with transitions:

$$\begin{array}{ll} 0 \xrightarrow{\langle A\rangle,\bot/\bot} 1 & 0 \xrightarrow{\langle A\rangle,\bot/\top} 2 \\ 1 \xrightarrow{\langle/A\rangle,\top/\bot} 3 & 1 \xrightarrow{\langle/A\rangle,\bot/\bot} 0 \\ 2 \xrightarrow{\langle/A\rangle,\top/\bot} 0 & 2 \xrightarrow{\langle/A\rangle,\bot/\bot} 3 \end{array}$$

  where 0 is the initial and accepting state. Note that this transducer is not sequential, and that 3 is a sink state.

Finally, we extend the definitions to non-named formulae:

– $\mathcal{B}[\![\mathsf{left}(\phi,\psi)]\!]$ is defined to be $\langle \mathrm{id}, \langle \mathcal{E}[\![\phi]\!], \mathcal{E}[\![\psi]\!]\rangle\rangle ; \mathcal{B}[\![\mathsf{left}]\!]$ where we define:

$$\mathcal{B}[\![\mathsf{left}]\!] = \mathcal{B}[\![\mathsf{left}, A_1]\!] \vee \cdots \vee \mathcal{B}[\![\mathsf{left}, A_n]\!]$$

for $\Sigma = \{A_1, \ldots, A_n\}$. The other modalities are defined similarly.

It is routine to verify that $\mathcal{B}[\![\phi]\!]$ and $\mathcal{E}[\![\phi]\!]$ satisfy the required properties. □

A.2 Separation and Proofs of Completeness for $\mathcal{X}_{\mathbf{until}}$

We will work here towards the proofs of Theorem 3 and Theorem 7. These results will all rely on the notion of separating into formulae that look in only one direction. We will need to define a few more of these "single-time fragments".

Definition 15 (Strict forward $\mathcal{X}_{\mathrm{until}}$). *Forward $\mathcal{X}_{\mathrm{until}}$ is the fragment of $\mathcal{X}_{\mathrm{until}}$ in which:*

– *all occurrences of* up *are of the form* $\mathsf{up}(\phi,\psi)$*, where ϕ and ψ have no top-level occurrences of* down*, and*
– *all occurrences of* left *are of the form* $\mathsf{down}(\phi \wedge \neg\langle\mathsf{left}\rangle\top, \psi)$*.*

A formula is in strict forward $\mathcal{X}_{\mathrm{until}}$ if it is in forward $\mathcal{X}_{\mathrm{until}}$ and has no top-level occurrences of down*.*

Strict forward formulas are those that only "look into the future". We also need formulas that look into the present:

Definition 16 (Stationary $\mathcal{X}_{\mathrm{until}}$). *Stationary $\mathcal{X}_{\mathrm{until}}$ is the fragment of $\mathcal{X}_{\mathrm{until}}$ in which there are no occurrences of any modalities.*

Our main result towards the completeness theorems, Theorem 3 and Theorem 7, is a separation theorem.

Theorem 10. *For every $\mathcal{X}_{\mathrm{until}}$ formula ϕ over labels Σ, there is a $\mathcal{X}_{\mathrm{until}}$ formula ψ over labels Σ which agrees with ϕ on any node of any un-nested ordered tree with labels Σ, such that ψ is a boolean combination of oscillation-free formulas that are either strict backward, strict forward, or backward downward.*

This result is inspired by a separation theorem stated by Marx in [17]. That paper states a coarser separation, which does not distinguish strict backward from strict forward. Such a distinction is not possible in the setting of Marx's paper, which works over general (not necessarily un-nested trees). In addition, his argument contains a mistake – his Lemma 10 includes the distributivity property:

$$\mathsf{down}(\phi, \psi \wedge \chi) = \mathsf{down}(\phi,\psi) \wedge \mathsf{down}(\phi,\chi)$$

which is only true when $\xrightarrow{\mathsf{down}}$ is deterministic. As a result, his induction (in an un-numbered "final step" at the end of Section 4) fails.

We do not know of a general solution to this problem, but in the case of un-nested trees, we can show that formulae $\mathsf{down}(\phi,\psi)$ are only needed in the case where ψ is stationary and so contains no up modalities to pull out.

$$\pi?(\phi,\psi) = \phi \vee (\psi \wedge \pi(\phi,\psi))$$

Fig. 1. Abbreviations used in the rewrite rules

$$\pi(\phi \vee \phi', \psi) = \pi(\phi,\psi) \vee \pi(\phi',\psi) \tag{1}$$

$$\begin{aligned}\mathsf{up}(\phi \wedge \mathsf{down}(\psi,\psi'),\phi') = \mathsf{up}(&\mathsf{left}(\mathsf{down}?(\psi,\psi'),\top) \wedge \mathsf{up}(\phi,\phi' \wedge \psi'),\phi') \\ \vee\ &\mathsf{left}(\mathsf{down}?(\psi,\psi'),\top) \wedge \mathsf{up}(\phi,\phi' \wedge \psi') \\ \vee\ &\mathsf{up}(\mathsf{right}(\mathsf{down}?(\psi,\psi'),\top) \wedge \mathsf{up}(\phi,\phi' \wedge \psi'),\phi') \\ \vee\ &\mathsf{right}(\mathsf{down}?(\psi,\psi'),\top) \wedge \mathsf{up}(\phi,\phi' \wedge \psi') \\ \vee\ &\mathsf{up}(\psi \wedge \mathsf{up}(\phi,\phi' \wedge \psi'),\phi') \\ \vee\ &\mathsf{up}(\phi,\phi' \wedge \psi') \wedge \mathsf{down}?(\psi,\psi')\end{aligned} \tag{2}$$

$$\begin{aligned}\mathsf{down}(\phi \wedge \mathsf{up}(\psi,\psi'),\phi') = \mathsf{down}(&\psi \wedge \mathsf{down}(\phi,\phi' \wedge \psi'),\phi') \\ \vee\ &(\mathsf{down}(\phi,\phi' \wedge \psi') \wedge \mathsf{up}?(\psi,\psi'))\end{aligned} \tag{3}$$

$$\mathsf{left}(\phi \wedge \mathsf{up}(\psi,\psi'),\phi') = \mathsf{left}(\phi,\phi') \wedge \mathsf{up}(\psi,\psi') \tag{4}$$

$$\mathsf{right}(\phi \wedge \mathsf{up}(\psi,\psi'),\phi') = \mathsf{right}(\phi,\phi') \wedge \mathsf{up}(\psi,\psi') \tag{5}$$

$$\mathsf{down}(\phi \wedge \mathsf{left}(\psi,\psi'),\phi') = \mathsf{down}(\psi \wedge \mathsf{right}(\phi,\psi'),\phi') \tag{6}$$

Fig. 2. Rewrite rules used in separation

$$\top = A_1 \vee \cdots \vee A_n \tag{7}$$

$$\pi(\phi,\psi \wedge \psi') = \pi(\phi,\psi) \wedge \pi(\phi,\psi') \quad (\text{for } \pi \neq \mathsf{down}) \tag{8}$$

$$\mathsf{left}(\phi,\psi) = \mathsf{left}(\phi,\psi) \wedge \mathsf{up}(\top,\bot) \tag{9}$$

$$\mathsf{right}(\phi,\psi) = \mathsf{right}(\phi,\psi) \wedge \mathsf{up}(\top,\bot) \tag{10}$$

$$\mathsf{up}(A \wedge \phi,\psi) = \mathsf{up}(A \wedge \phi,\top) \wedge \neg\mathsf{up}(\mathsf{up}(A,\top) \wedge \neg\psi,\top) \tag{11}$$

$$\neg\mathsf{up}(A \wedge \phi,\psi) = \mathsf{up}(A \wedge \neg\phi,\psi) \vee \neg\mathsf{up}(A,\psi) \tag{12}$$

$$A \wedge \mathsf{down}(\phi,\psi) = A \wedge \mathsf{down}(\phi \wedge \mathsf{up}(A,\psi),\top) \tag{13}$$

$$\mathsf{down}(\phi \wedge \neg\mathsf{up}(\psi,\top),\phi') = \mathsf{down}(\phi,\phi' \wedge \neg\psi) \wedge \neg\mathsf{up}(\psi,\top) \wedge \neg\psi \tag{14}$$

$$\mathsf{left}(\phi \wedge \neg\mathsf{up}(\psi,\psi'),\phi') = \mathsf{left}(\phi,\phi') \wedge \neg\mathsf{up}(\psi,\psi') \tag{15}$$

$$\mathsf{right}(\phi \wedge \neg\mathsf{up}(\psi,\psi'),\phi') = \mathsf{right}(\phi,\phi') \wedge \neg\mathsf{up}(\psi,\psi') \tag{16}$$

$$\mathsf{left}(\phi,\phi' \vee \mathsf{up}(\psi,\psi')) = \mathsf{left}(\phi,\phi') \vee (\mathsf{left}(\phi,\top) \wedge \mathsf{up}(\psi,\psi')) \tag{17}$$

$$\mathsf{left}(\phi,\phi' \vee \neg\mathsf{up}(\psi,\psi')) = \mathsf{left}(\phi,\phi') \vee (\mathsf{left}(\phi,\top) \wedge \neg\mathsf{up}(\psi,\psi')) \tag{18}$$

$$\mathsf{right}(\phi,\phi' \vee \mathsf{up}(\psi,\psi')) = \mathsf{right}(\phi,\phi') \vee (\mathsf{right}(\phi,\top) \wedge \mathsf{up}(\psi,\psi')) \tag{19}$$

$$\mathsf{right}(\phi,\phi' \vee \neg\mathsf{up}(\psi,\psi')) = \mathsf{right}(\phi,\phi') \vee (\mathsf{right}(\phi,\top) \wedge \neg\mathsf{up}(\psi,\psi')) \tag{20}$$

$$\mathsf{down}(\phi,\psi) = \mathsf{down}(\mathsf{left}?(\phi,\top) \wedge \neg\mathsf{right}(\top,\bot),\psi) \tag{21}$$

$$\mathsf{down}(\phi,\psi) = \mathsf{down}(\mathsf{right}?(\phi,\top) \wedge \neg\mathsf{left}(\top,\bot),\psi) \tag{22}$$

Fig. 3. Additional rewrite rules used in separation of $\mathcal{X}_{\text{until}}$, where $\Sigma = \{A_1,\ldots,A_n\}$

Proof (of Theorem 10). We use the rewrite rules in Figures 2 and 3 plus the usual De Morgan and distributive properties of boolean algebra, and the rules needed to perform separation in LTL [10] (applied to left and right), to rewrite ϕ.

Define a formula to be *separated* whenever it is a boolean combination of oscillation-free formulae that are either strict backward, strict forward, or downward. Define a formula to be *vertically named* whenever:

- all occurrences of down are of the form $A \wedge \mathsf{down}(\phi, \psi)$, and
- all occurrences of up are of the form $\mathsf{up}(A \wedge \phi, \psi)$.

Define a formula to be *vertically separated* whenever:

1. every occurrence of up or down is of the form $\pi(\phi, \psi)$ where ψ is stationary,
2. ever subformula of the form $\mathsf{up}(\phi, \psi)$ has no top-level occurrences of down in ϕ or ψ, and
3. every subformula of the form $\pi(\phi, \psi)$ with $\pi \neq \mathsf{up}$ has no occurrences of up in ϕ or ψ.

The outline of the proof is:

I. First show that every formula can be rewritten to a named formula, and moreover that this rewriting preserves properties 1–2.
II. Show that every downward formula satisfying property 1 can be rewritten to a backward downward formula, and also to a forward downward formula. This will be used in part IV.
III. Next, show that every named formula can be rewritten to a vertically named, vertically separated formula. This will make use of part I.
IV. Finally, show that every vertically named, vertically separated formula can be rewritten to a separated formula. This will make use of part II.

Given this outline, the final proof will proceed by applying part I to get a named formula, then applying part III to get a vertically named, vertically separated formula, and finally applying part IV to get a separated formula.

We first show I – i.e. that every $\mathcal{X}_{\text{until}}$ formula ϕ can be rewritten to a named $\mathcal{X}_{\text{until}}$ formula ϕ', and moreover that this rewriting preserves properties 1–2 in the definition of vertically separated. This is a simple induction over ϕ, for which the interesting cases are:

- If $\phi = \mathsf{up}(\psi, \psi')$ then by induction we can assume ψ and ψ' to be named. Then by Rules 1 and 7 we have:

$$\phi = \mathsf{up}(A_1 \wedge \psi, \psi') \vee \cdots \vee \mathsf{up}(A_n \wedge \psi, \psi')$$

  It is routine to check that this is named, and satisfies properties 1–2 whenever ϕ does. The case of down is handled similarly.
- If $\phi = \mathsf{left}(\psi, \psi')$ then by induction we can assume ψ and ψ' to be named. Then by Rules 1, 9, and 7 we have:

$$\phi = (\langle \mathsf{up} \rangle A_1 \wedge \mathsf{left}(\psi, \psi')) \vee \cdots \vee (\langle \mathsf{up} \rangle A_n \wedge \mathsf{left}(\wedge \psi, \psi'))$$

It is routine to check that this is named, and satisfies properties 1–2 whenever ϕ does. The case of right is handled similarly.

We next show II – every downward $\mathcal{X}_{\text{until}}$ formula ϕ satisfying property 1 can be rewritten to a formula in backward downward $\mathcal{X}_{\text{until}}$ (or dually, forward downward $\mathcal{X}_{\text{until}}$). This is an induction on ϕ, for which the interesting case is:

– If $\phi = \mathsf{down}(\psi, \psi')$ then, since ϕ satisfies property 1, we have that ψ' is stationary. We can regard ψ as an LTL formula with until on left and right, and atomic properties which are downward formulae. We can then appeal to separation of LTL to find a right-free χ over the same atomic properties such that:

$$\mathsf{left?}(\psi, \top) \wedge \neg\mathsf{right}(\top, \bot) = \chi \wedge \neg\mathsf{right}(\top, \bot)$$

By induction, we can find a formula χ' equivalent to χ, where each downward formula in χ has been replaced by an equivalent backward downward formula. We then use Rule 21 to get:

$$\phi = \mathsf{down}(\chi \wedge \neg\mathsf{right}(\top, \bot), \psi')$$

as required.

We now turn to showing part III – every named $\mathcal{X}_{\text{until}}$ formula ϕ can be rewritten to a vertically separated formula:

– We prove by induction on ϕ that every named $\mathcal{X}_{\text{until}}$ formula can be rewritten to a formula satisfying property 1, for which the interesting cases are:
  - If $\phi = \mathsf{up}(A \wedge \psi, \psi')$ then by induction we can assume ψ and ψ' to satisfy property 1. We then use Rule 11:

    $$\phi = \mathsf{up}(A \wedge \psi, \top) \wedge \neg\mathsf{up}(\mathsf{up}(A, \top) \wedge \neg\psi', \top)$$

    It is routine to check that this satisfies property 1.
  - If $\phi = A \wedge \mathsf{down}(\psi, \psi')$ then by induction we can assume ψ and ψ' to satisfy property 1. We then use Rule 13:

    $$\phi = A \wedge \mathsf{down}(\psi \wedge \mathsf{up}(A, \psi'), \top)$$

    then use the previous case to rewrite $\mathsf{up}(A, \psi')$.

  In the process above, we may lose the property of being named, but we can regain it by applying part II.
– We prove by induction on ϕ that every named $\mathcal{X}_{\text{until}}$ formula satisfying property 1 can be rewritten to a formula satisfying properties 1 and 2, for which the interesting case is when $\phi = \mathsf{up}(A \wedge \psi, \psi')$. By induction we can assume ψ and ψ' to satisfy properties 1 and 2. Since ϕ satisfies property 1, ψ' is stationary, and so contains no occurrences of up. Without loss of generality, we can assume ψ to be in disjunctive normal form, and we proceed by induction on the number of occurrences of down in ψ:

- If ψ contains no top-level occurrences of down, then ϕ satisfies properties 1 and 2.
- If $\psi = \psi'' \vee (\psi''' \wedge \mathsf{down}(\chi, \chi'))$ then we have by Rules 1 and 2:

$$\begin{aligned}\phi = \mathsf{up}(&A \wedge \psi'', \psi') \\ &\vee \mathsf{up}(\mathsf{left}(\mathsf{down}?(\chi, \chi'), \top) \wedge \mathsf{up}(A \wedge \psi''', \psi' \wedge \chi'), \psi') \\ &\vee \mathsf{left}(\mathsf{down}?(\chi, \chi'), \top) \wedge \mathsf{up}(A \wedge \psi''', \psi' \wedge \chi') \\ &\vee \mathsf{up}(\mathsf{right}(\mathsf{down}?(\chi, \chi'), \top) \wedge \mathsf{up}(A \wedge \psi''', \psi' \wedge \chi'), \psi') \\ &\vee \mathsf{right}(\mathsf{down}?(\chi, \chi'), \top) \wedge \mathsf{up}(A \wedge \psi''', \psi' \wedge \chi') \\ &\vee \mathsf{up}(\chi \wedge \mathsf{up}(A \wedge \psi''', \psi' \wedge \chi'), \psi') \\ &\vee \mathsf{up}(A \wedge \psi''', \psi' \wedge \chi') \wedge \mathsf{down}?(\chi, \chi')\end{aligned}$$

  and we proceed by induction.
- If $\psi = \psi'' \vee (\psi''' \wedge \neg\mathsf{down}(\chi, \chi'))$ then we have by Rules 1 and 12:

$$\begin{aligned}\phi = \mathsf{up}(&A \wedge \psi'', \psi') \\ &\vee \mathsf{up}(A \wedge \psi''', \psi') \wedge \neg\mathsf{up}(A \wedge \mathsf{down}(\chi, \chi'), \psi')\end{aligned}$$

  and we proceed using the previous case.

In the process above, we may lose the property of being named, but we can regain it by applying part II.

– We prove by induction on ϕ that every named $\mathcal{X}_{\text{until}}$ formula satisfying properties 1 and 2 can be rewritten to a vertically named, vertically separated formula. This is similar to the previous induction, making use of Rules 3–5 for positive occurrences of up, and Rules 14–20 for negative occurrences.

This completes the proof of part III.

Finally, we show part IV – every vertically named, vertically separated $\mathcal{X}_{\text{until}}$ formula ϕ can be rewritten to a separated formula. This is a simple induction over ϕ, for which the interesting cases are:

– If $\phi = \mathsf{up}(A \wedge \psi, \psi')$ then by induction we can assume ψ and ψ' are separated. Since ϕ is vertically separated, we have that ψ' is stationary. Without loss of generality, we can assume ψ is in disjunctive normal form, and use rule 1 to distribute until through disjunction, so the interesting case is when:

$$\phi = \mathsf{up}(A \wedge \psi_{\mathsf{left}} \wedge \psi_{\mathsf{right}} \wedge \psi_{\mathsf{down}}, \psi')$$

  where ψ_{left} is strict backward, ψ_{right} is strict forward, and ψ_{down} is backward downward. Since ϕ is vertically separated, ψ_{down} must be stationary, and so we use Rules 1 and 12 to get:

$$\phi = \mathsf{up}(A \wedge \psi_{\mathsf{left}}, \psi') \wedge \mathsf{up}(A \wedge \psi_{\mathsf{right}}, \psi') \wedge \mathsf{up}(A \wedge \psi_{\mathsf{down}}, \psi')$$

  which is separated, as required.
– If $\phi = A \wedge \mathsf{down}(\psi, \psi')$ then, since ϕ is vertically separated, we have that ϕ contains no occurrences of up, and so ϕ is downward. By part II, downward formulae can be rewritten to backward downward formulae.

– If $\phi = \mathsf{left}(\psi, \psi')$ or $\mathsf{right}(\psi, \psi')$ then, since ϕ is vertically separated, we can regard ϕ as an LTL formula with until on left and right, and atomic properties which are downward formulae. We can then appeal to separation of LTL to find ϕ' which is a boolean combination of:
  - LTL formulae χ containing only until on left, with atomic properties which are downward formulae. By part II we can rewrite those atomic properties to be backward downward formulae, and hence rewrite χ to be a boolean combination of formulae that are either strict backward or downward.
  - LTL formulae χ containing only until on right, which are handled similarly, getting a boolean combination of formulae that are either strict forward or forward.

This completes the proof of Theorem 10, modulo the verification that that the rewrites in Figures 2 and 3 are valid for un-nested trees, which is routine. □

Note that an alternative proof technique would be to use the fact that un-nested trees with alphabet of size N have depth bounded by N, and that for such trees until can be translated to $\langle\pi\rangle$:

$$\pi(\phi, \psi) = \pi^{\leq N}(\phi, \psi)$$

where we define:

$$\pi^{\leq 0}(\phi, \psi) = \bot \qquad \pi^{\leq i+1}(\phi, \psi) = \langle\pi\rangle\phi \vee \langle\pi\rangle(\psi \wedge \pi^{\leq i}(\phi, \psi))$$

The rewrite rules can then be specialized to $\langle\pi\rangle$. The reasons for using the proof given here are:

– the use of naming, rather than bounded depth, fits better with the use of naming in constructing transducer networks, and
– the rewrite rules given in Figures 2 and 3 make it clear where we are relying on un-nested documents: Rules 11, 12 and 13 are the only ones which rely on un-nested documents, and Rule 7 is the only one which relies on a fixed, finite alphabet.

Theorem 3 follows from Theorem 10 and the proposition below, whose proof is straightforward; it implies that if we consider formulas at the root, then we need only the downward component in separation.

Proposition 6. *If ϕ is in strict forward or strict backward $\mathcal{X}_{\text{until}}$, then there is a stationary formula ψ equivalent to ϕ at the root of any tree.*

The proposition is straightforward; at the root we cannot go up, hence any subformula of an oscillation-free strict forward or strict backward formula that uses upward modalities will be equivalent to true or false.

We say that two formulas are *begin-equivalent* if they agree on any nodes that are on the far bottom right of a tree: that is, nodes which have no children, no right siblings, and whose ancestors all have no right siblings. We say that they are *end-equivalent* if they agree on any nodes that are on the right of a tree: that

is, nodes which have no right siblings, and for whose ancestors all have no right siblings.

Note that if two formulas are begin-tag determined, then begin-equivalence implies equivalence over any node in any tree. Similarly, if two formulas are end-tag determined, then end-equivalence implies equivalence. Note also that strict backward formulas are clearly begin-tag determined, and backward formulas are end-tag determined. Using this observation, Theorem 7 thus follows from Theorem 10 and the following simple result, analogous to Proposition 6. It says that the forward parts of a separated formula can be eliminated, assuming that we are only interested in end-equivalence, and that the forward and downward parts can be eliminated if we are only interested in begin-equivalence.

Proposition 7. *Every $\mathcal{X}_{\text{until}}$ formula ϕ that is oscillation-free and strict forward (resp. downward or strict forward) is end-equivalent (resp. begin-equivalent) to an oscillation-free strict backward formula.*

The proposition in the case of end-equivalence follows from the fact that for nodes on the right of a tree, one can not make a rightward move from an ancestor. Hence all but upward modalities within a strict forward formula can be eliminated, and we are left with a strict backward formula; indeed, we are left with a formula that uses only upward modalities. In the case of begin-equivalence, we additionally note that for a node on the bottom right all modalities can be eliminated from a downward formula.

A.3 Proofs of Completeness for +HML

For the results on +HML, Theorem 4, and Theorem 8, there is a simpler argument, a variant of that used in [20] and [4]. The argument below is particularly close to that of Theorem 5.1 of [4], adapted to ordered trees. We translate +HML to logical formulas, and then show that these formulas can be normalized to be of a special form. This normal form is a variant of the "tree pattern queries" of [4]. The ability to rewrite a +HML formula into the normal form can be thought of as an analog of separation (Theorem 10) for +HML. Given a normalized formula, we can apply root-equivalence, end-equivalence, or begin-equivalence to the normalized formula, arriving at a logical formula in which all the bound variables are restricted to lie in a certain relation to the free variable. Finally, we translate the syntactic restrictions back into +HML.

We will now give more detail, leaving some of the proofs to the reader, since they are similar to those in [4]. Let $\exists^+FO$ denote "positive existential first-order logic"– the fragment of first-order logic built up using just $\wedge, \vee, \exists$, over the vocabulary with unary predicates $A(x)$ for $A \in \Sigma$ and binary "axis relations" $\mathsf{right}(x,y), \mathsf{right}^+(x,y), \mathsf{down}(x,y), \mathsf{down}^+(x,y)$ for the rightward and downward axes. The atomic predicates above have the obvious interpretation in ordered trees. Formalizing the semantics of HML within first-order logic, we see that every +HML formula is equivalent to an $\exists^+FO$ formula, which can be assumed

to be in prefix normal form. In particular, we can translate +HML formulas into $\exists^+FO$ formulas of the form

$$\rho(x_0) = \exists x_1 \ldots \exists x_m \bigvee_{i<n} \phi_i(x_0, x_1, \ldots, x_m)$$

where ϕ_i is a conjunction of equalities, unary predicates, and axis relations mentioning $x_1 \ldots x_m$ and the free variable x_0.

A *positive atomic type* (or just "type" for short) is a conjunction of atomic formulas that is consistent and complete: that is, there is some tree T and nodes $n_0 \ldots n_m$ in T that satisfy the conjunction, and such the conjunction contains all atomic formulas that hold between $n_0 \ldots n_m$. Note that if this holds for some T and nodes $n_0 \ldots n_m$, then it holds for all such T and $n_0 \ldots n_m$

Given a type γ, and variables x_i, x_j we will say that x_i is a descendant of x_j in γ if there is a conjunct of γ that asserts this, and similarly for the other axis relations and unary predicates. A type γ is called *lub-closed* if whenever we have distinct variables x_i, x_j that are not related by an axis relation, there are variables x_l and x_m (not necessarily distinct from x_i or from x_j), such that x_i is a descendant of x_l, x_j is a descendant of x_m, and x_l is related to x_m by one of the sibling relations.

The following is easy to show:

Proposition 8. *Every* +HML *formula* $\rho(x_0)$ *is equivalent to one of the form:*

$$\exists x_1 \ldots \exists x_m \bigvee_{i<n} \phi_i(x_0, x_1 \ldots x_n)$$

where each ϕ_i *is a lub-closed type.*

We now give the analogs of strict backward and backward in the setting of logic. A type is *forward-free* if no variable x_i is a right sibling of x_0 and no variable x_i is a right sibling of a variable that is an ancestor of x_0. It is *downward-free* if no variable x_i is a descendant of x_0.

Proposition 9. *Every* $\exists^+FO$ *formula* $\rho(x_0)$ *of the form*

$$\exists x_1 \ldots \exists x_m \bigvee_{i<n} \phi_i$$

is end-equivalent (resp. begin-equivalent) to one of the same form in which each ϕ_i *is forward-free (resp. forward-free and downward-free) Every formula of the above form is root-equivalent to one where every variable is a descendant of* x_0 *or equal to* x_0*.*

Proof. By the previous proposition, we can take each ϕ_i to be a lub-closed type. The lub-closed property implies that every bound variable is either a descendant of the free variable x_0, equal to the free variable x_0, an ancestor of x_0, a descendant of a left sibling of an ancestor of x_0, or a descendant of a right sibling of an ancestor of x_0. But for end-equivalence, we restrict to trees on which there are no

nodes that are right siblings of an ancestor of x_0, and thus we can eliminate any type that includes these variables from the disjunction. The resulting formula is forward-free. In the case of begin-equivalence, we can likewise eliminate types that have variables that are either right siblings of an ancestor of x_0 or below x_0.

Finally, we convert back into +HML:

Proposition 10. *Every formula of the form*

$$\exists x_1 \ldots \exists x_m \bigvee_{i<n} \phi_i$$

where each ϕ_i is a forward-free (resp. forward-free and downward-free) type, is equivalent to an oscillation-free formula in backwards +HML *(resp. strict backwards* +HML*).*

Every formula in which every variable is a descendant of x_0 or equal to x_0 is equivalent to one in downwards +HML*.*

The proof of the proposition is by induction on the number of bound variables, and follows the argument for going from tree patterns into XPath in [4] (Part 1. of Theorem 3.2).

Theorem 4, and Theorem 8, now follow from Proposition 8, Proposition 9, and Proposition 10. In the case of Theorem 8, we use also the fact that for begin-tag (resp. end-tag) determined queries, begin-equivalence (resp. end-equivalence) implies equivalence at all nodes.

A.4 Other Proofs

The remaining results to prove are:

- Proposition 2, which follows from the example formulae ϕ_n in Section 3.
- Proposition 3, which follows from letting $\phi = \langle \mathsf{right}^+ \rangle A$, and considering input documents of the form $\langle C \rangle \langle B/ \rangle^n \langle A/ \rangle \langle /C \rangle$, which can only produce one token of the output $(\langle C \rangle, \bot)(\langle B/ \rangle, \top)^n (\langle A/ \rangle, \bot)(\langle /C \rangle, \bot)$ before the $\langle A \rangle$ input is reached, and
- Theorem 5, which we prove below.

Our proof of Theorem 5 will use the following variant of the standard notion of bisimulation:

Definition 17 (Stratified Bisimulation). *Given two ordered trees T and T', a stratified bisimulation over Π is a family of relations $R_i \subseteq (N \times N') : i \in \omega$ such that, for any $(n, n') \in R_{i+1}$ and $\ell \in \Pi$ we have:*

- $\lambda(n) = \lambda(n')$,
- *for all $n \xrightarrow{\ell} m$ in T there exists $n' \xrightarrow{\ell} m'$ in T' with $(m, m') \in R_i$, and*
- *for all $n' \xrightarrow{\ell} m'$ in T' there exists $n \xrightarrow{\ell} m$ in T with $(m, m') \in R_i$.*

The connection of bisimulation to inexpressivity results is as follows:

Proposition 11. *For any* HML *formula ϕ of size k, and for any ordered trees T and T' with stratified bisimulation $R_i : i \in \omega$ over the modalities contained in ϕ, we have $(n, n') \in R_k$ implies $T, n \vDash \phi$ precisely when $T', n' \vDash \phi$.*

Theorem 5 follows immediately from the following:

Proposition 12. *The* HML *formula $\langle \mathsf{down} \rangle (B \wedge \neg \langle \mathsf{right}^+ \rangle A)$ is not equivalent to any formula in backward* HML.

Proof. We will show something stronger: namely that the formula above is not equivalent to any formula whose only use of the rightward axes is of the form $\langle \text{downlast} \rangle \phi$, defined to be:

$$\langle \text{downlast} \rangle \phi = \langle \mathsf{down} \rangle (\phi \wedge \neg \langle \mathsf{right} \rangle \top)$$

that is, $\xrightarrow{\text{downlast}}$ is the relation defined:

$$n' \xrightarrow{\text{downlast}} m \text{ whenever } n \xrightarrow{\mathsf{down}} m \not\xrightarrow{\mathsf{right}}$$

Define the set of modalities Π as:

$$\Pi = \{\mathsf{down}, \mathsf{down}^+, \mathsf{left}, \mathsf{left}^+, \mathsf{up}, \mathsf{up}^+, \text{downlast}\}$$

We will show that for any HML formula ϕ of size K containing only modalities in Π we can find trees T and T' which are distinguished by $\langle \mathsf{down} \rangle (B \wedge \neg \langle \mathsf{right}^+ \rangle A)$ but not by ϕ, from which the result follows. The trees T and T' are given as:

$$\begin{aligned} \text{stream}(T) &= \langle R \rangle (\langle A/ \rangle \langle C/ \rangle^{K-1} \langle B/ \rangle \langle C/ \rangle^{K-1})^K \langle A/ \rangle \langle C/ \rangle^{K-1} \langle /R \rangle \\ \text{stream}(T') &= \langle R \rangle (\langle A/ \rangle \langle C/ \rangle^{K-1} \langle B/ \rangle \langle C/ \rangle^{K-1})^K \langle /R \rangle \end{aligned}$$

that is T is given by:

$$\begin{gathered} n_0 \xrightarrow{\mathsf{down}} n_1 \xrightarrow{\mathsf{right}} \cdots \xrightarrow{\mathsf{right}} n_{2K^2+K} \\ \lambda(n_0) = R \quad \lambda(n_1, \ldots, n_{2K^2+K}) = (AC^{K-1}BC^{K-1})^K AC^{K-1} \end{gathered}$$

and T' is given by:

$$\begin{gathered} n'_0 \xrightarrow{\mathsf{down}} n'_1 \xrightarrow{\mathsf{right}} \cdots \xrightarrow{\mathsf{right}} n'_{2K^2} \\ \lambda(n'_0) = R \quad \lambda(n'_1, \ldots, n'_{2K^2}) = (AC^{K-1}BC^{K-1})^K \end{gathered}$$

Clearly the formula $\langle \mathsf{down} \rangle (B \wedge \neg \langle \mathsf{right}^+ \rangle A)$ is false at the root of T and true at the root of T'. We will now construct a stratified bisimulation over Π containing $(n_0, n'_0) \in R_K$, and so from Proposition 11, ϕ cannot distinguish T from T'. Our stratified bisimulation is:

$$\begin{aligned} &\text{for all } i > K \quad R_i = \emptyset \\ &\text{for all } i \leq K \quad R_i = \{(n_j, n'_j) \mid j \leq 2K^2\} \\ &\qquad\qquad \cup \{(n_{2K^2+j}, n'_{2K^2+j-K}) \mid i < j \leq K\} \\ &\qquad\qquad \cup \{(n_{2Kk+j}, n'_{2Kk+j-2K}) \mid i < k \leq K, j \leq K\} \end{aligned}$$

It is routine to verify that this satisfies the conditions of a stratified bisimulation, and so the result holds. □