A Direct Translation from XPath to Nondeterministic Automata

Nadime Francis[1], Claire David[2], and Leonid Libkin[3]

[1] ENS-Cachan
[2] Univ Paris-Est Marne-la-Vallée
[3] Univ of Edinburgh

Abstract. Since navigational aspects of XPath correspond to first-order definability, it has been proposed to use the analogy with the very successful technique of translating LTL into automata, and produce efficient translations of XPath queries into automata on unranked trees. These translations can then be used for a variety of reasoning tasks such as XPath consistency, or optimization, under XML schema constraints. In the verification scenarios, translations into both nondeterministic and alternating automata are used. But while a direct translation from XPath into alternating automata is known, only an indirect translation into nondeterministic automata - going via intermediate logics - exists. A direct translation is desirable as most XML specifications have particularly nice translations into nondeterministic automata and it is natural to use such automata to reason about XPath and schemas. The goal of the paper is to produce such a direct translation of XPath into nondeterministic automata.

1 Introduction

XML reasoning tasks have been addressed in many recent papers. Typical problems include consistency of type declarations and constraints [1, 2, 16], or of schema specifications and navigational properties [3, 11] or containment of XPath expressions [4, 5, 10, 18, 26]. Static analysis tasks are often addressed using logic and automata-based techniques [6]. Thus, due to the well-known correspondences between XML formalisms and logics and automata on trees, it is natural to apply logic and automata techniques to reasoning about XML.

Indeed, several attempts have been made to find generic logic/automata tools applicable in a variety of XML reasoning tasks. The idea of such approaches is roughly as follows. Suppose the task at hand is the well-known XPath containment problem under DTDs: given a DTD d and two XPath expressions e_1 and e_2, is it true that $d \models e_1 \subseteq e_2$? In other words, is it true that for each XML tree T that conforms to d, the expression e_1 selects a subset of nodes selected by e_2? Suppose we can somehow capture by an automaton $\mathcal{A}_e$ the set of nodes selected by the expression $e_1 \wedge \neg e_2$, i.e., counterexamples to containment. Schemas such as DTDs, Relax NG, and others, are easily translated into tree automata (in fact most designs of schemas for XML are based on tree automata). So assume that we have an automaton $\mathcal{A}_d$ that accepts trees that conform to d. Then to see whether $d \models e_1 \subseteq e_2$ one simply checks whether

$$\mathcal{L}(\mathcal{A}_d \times \mathcal{A}_e) \neq \emptyset$$

since a tree in $\mathcal{L}(\mathcal{A}_d \times \mathcal{A}_e)$ is a counterexample to $d \models e_1 \subseteq e_2$. Throughout the paper, $\mathcal{L}(\mathcal{A})$ is the language accepted by automaton $\mathcal{A}$.

This is essentially the same as the main idea of automata-based verification [6], whose goal is to check whether a program P satisfies a specification φ. One views an abstraction of P as an automaton $\mathcal{A}_P$, constructs an automaton $\mathcal{A}_{\neg\varphi}$ capturing counterexamples to the specification (i.e., program executions that do not satisfy φ), and checks for nonemptiness of $\mathcal{L}(\mathcal{A}_P \times \mathcal{A}_{\neg\varphi})$. If the language is nonempty, it contains counterexamples to the specification, i.e., bugs in P.

In the verification context, one most commonly deals with automata over ω-words, as they properly capture traces of program behavior and specifications. In the XML context, we deal with queries on finite trees, and finite (unranked) tree automata. The idea of adapting verification techniques to static analysis of XML is therefore very natural and it has been explored in a number of papers [5, 7, 10, 11, 13, 15, 18].

In the classical verification scenario, the specification logic is most often LTL (linear temporal logic), which happens to have the same expressive power as first-order logic (FO). In our context, we deal with XPath, whose navigational capabilities are expressible in FO and which, with an addition of a certain conditional construct, captures FO [19]. If we want to adapt existing LTL-to-automata translations, there are two main approaches to it: translating formulae into nondeterministic, or alternating automata [30, 29]. The difference between these approaches is where the complexity lies – in the translation itself, or in checking nonemptiness of languages defined by automata:

Automata / Task	Nondeterministic	Alternating
Translation	exponential	polynomial
Nonemptiness checking	polynomial	exponential

Both techniques have their advantages; the choice is really about the task to which algorithmic and optimization techniques will be applied. In the XML context, there are at least two reasons to seriously consider translating into nondeterministic automata. First, most schema formalisms come from nondeterministic tree automata, so these translations appear to be better suited for reasoning tasks involving schemas. Second, one may hope to use a rich arsenal of optimization tools developed for the original Vardi-Wolper translation [30] of LTL into nondeterministic Büchi automata [8, 30, 12] and adapt it for XML reasoning tasks.

Both approaches have been explored in the XML context. In [5], a direct translation from XPath to a certain type of alternating automata on unranked trees is given. In [15], translations into nondeterministic automata are explored, but the logic used for translation is not XPath, but rather an LTL-like logic on trees, first introduced in [25] and used later in [19]. While the translation is single-exponential, this is not satisfactory since the translation from XPath into that logic is single-exponential itself! Curiously enough, this does not make the whole translation from XPath into automata double-exponential, but showing this requires a careful analysis of the structure of subformulas of formulas which are translations of XPath queries. However, this intermediate step, with its exponential blowup, seems to preclude any possibility of providing efficient implementations of XPath-to-nondeterministic automata translations, as very little of the original XPath query can still be seen and used in its translation.

Thus, we would like to have a *direct translation from XPath queries into nondeterministic tree automata*. This is the goal of this paper. We provide such a translation; in fact, we do a bit more. Since XPath node queries select nodes from documents, we provide a translation into *query automata* [20, 22] which not only accept or reject trees, but also select nodes from them. For example, for the task of checking containment of queries, this will give us not just a yes/no answer, but a concise description of *all* counterexamples to containment.

The translation will be single-exponential: in case of nondeterministic automata, one cannot do better than this, as LTL fragments requiring this complexity are easily coded in XPath. As the exact language, we use Conditional XPath of [19], which is FO-complete; it extends the standard XPath with an analog of the temporal 'until' operator.

The translation follows the spirit of the Vardi-Wolper translation, and thus one can expect that optimization techniques developed for the latter (e.g., those in [8, 30, 12]) can be adapted for our construction.

Organization In Section 2 we describe XML documents (unranked trees), automata for them, and query automata. In Section 3 we give the syntax and semantics of (conditional) XPath. The translation from XPath to query automata is given in Section 4. Its correctness is shown in Section 5.

2 Preliminaries

XML documents as unranked trees XML documents are normally abstracted as labeled unranked trees [21, 14, 27]. Nodes in unranked trees are elements of $\mathbb{N}^*$, i.e. strings of natural numbers. We write $s \cdot s'$ for the concatenation of strings, and ε for the empty string. The basic binary relations on $\mathbb{N}^*$ are:

- the *child relation*: $s \prec_{\text{ch}} s'$ if $s' = s \cdot i$, for some $i \in \mathbb{N}$, and
- the *next-sibling* relation: $s' \prec_{\text{ns}} s''$ if $s' = s \cdot i$ and $s'' = s \cdot (i+1)$ for some $s \in \mathbb{N}^*$ and $i \in \mathbb{N}$.

The *descendant* relation $\prec^*_{\text{ch}}$ and the younger sibling relation $\prec^*_{\text{ns}}$ are the reflexive-transitive closures of $\prec_{\text{ch}}$ and $\prec_{\text{ns}}$.

An *unranked tree domain* D is a finite prefix-closed subset of $\mathbb{N}^*$ such that $s \cdot i \in D$ implies $s \cdot j \in D$ for all $j < i$. If Σ is a finite alphabet, a Σ-labeled *unranked tree* is a pair $T = (D, \lambda)$, where D is a tree domain and λ is a labeling function $\lambda : D \to \Sigma$.

Unranked tree automata A *nondeterministic unranked tree automaton* (cf. [21, 27]) over Σ-labeled trees is a triple $\mathcal{A} = (Q, F, \delta)$ where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and δ is a mapping $Q \times \Sigma \to 2^{Q^*}$ such that each $\delta(q, a)$ is a regular language over Q. We assume that each $\delta(q, a)$ is given as an NFA (nondeterministic finite automaton). A *run* of $\mathcal{A}$ on a tree $T = (D, \lambda)$ is a function $\rho_\mathcal{A} : D \to Q$ such that if $s \in D$ is a node with n children, and $\lambda(s) = a$, then the string $\rho_\mathcal{A}(s \cdot 0) \cdots \rho_\mathcal{A}(s \cdot (n-1))$ is in $\delta(\rho_\mathcal{A}(s), a)$. Thus, if s is a leaf labeled a, then $\rho_\mathcal{A}(s) = q$ implies that $\varepsilon \in \delta(q, a)$. A run is *accepting* if $\rho_\mathcal{A}(\varepsilon) \in F$, and a tree is *accepted* by $\mathcal{A}$ if an accepting run exists. Sets of trees accepted by automata $\mathcal{A}$ are called regular and denoted by $L(\mathcal{A})$.

Unranked tree automata are relevant in the context of *schemas* for XML document. There are multiple notions of *schemas*. What is common for such notions is that their structural aspects are subsumed by unranked tree automata, see [17] for several examples. More, translations from various schema formalisms into automata are usually very effective [17], and thus automata are naturally viewed as an abstraction of schemas in the XML literature. So when we speak of XML schemas, we shall assume that they are given by unranked tree automata.

Query automata It is well known that automata capture the expressiveness of MSO (monadic second order logic) sentences over finite and infinite strings and trees [28]. The model of query automata [22] captures the expressiveness of MSO formulae $\varphi(x)$ with one free first-order variable – that is, MSO-definable unary queries. We present here a nondeterministic version, as in [20, 9].

A *query automaton (QA)* for Σ-labeled unranked trees is a tuple $\mathcal{QA} = (Q, F, Q_s, \delta)$, where (Q, F, δ) is a nondeterministic unranked tree automaton, and $Q_s \subseteq Q$ is the set of *selecting states*. The runs of $\mathcal{QA}$ on a tree T are defined as the runs of (Q, F, δ) on T. Each run ρ of $\mathcal{QA}$ on a tree $T = (D, \lambda)$ defines the set $S_\rho(T) = \{s \in D \mid \rho(s) \in Q_s\}$ of nodes assigned to a selecting state. The unary query defined by $\mathcal{QA}$ is then, under the *existential semantics*,

$$\mathcal{QA}^\exists(T) \;=\; \bigcup\{S_\rho(T) \mid \rho \text{ is an accepting run of } \mathcal{QA} \text{ on } T\}.$$

Alternatively, one can define $\mathcal{QA}^\forall(T)$ under the *universal semantics* as $\bigcap\{S_\rho(T) \mid \rho \text{ is an accepting run of } \mathcal{QA} \text{ on } T\}$. Both semantics capture the class of unary MSO queries [20].

To eliminate the need to choose the semantics, and problems related to it (for example, the most natural constructing for the complement of a QA under the existential semantics produces a QA under the universal semantics), one can use *single-run query automata*. Such QA satisfy the conditions that for every tree T, and every two accepting runs ρ_1 and ρ_2 on T, we have $S_{\rho_1}(T) = S_{\rho_2}(T)$. For such QAs, we can unambiguously define the set of selected nodes as $\mathcal{QA}(T) = S_\rho(T)$, where ρ is an arbitrarily chosen accepting run.

This is not a restriction in terms of the expressiveness, as the following result is known:

Proposition 1. (see [9, 23, 24, 15]) *For every query automaton there exists an equivalent single-run query automaton.*

It is known that nonemptiness problem for QAs (single-run, or under the existential semantics) is solvable in polynomial time. Single-run QAs are easily closed under intersection: the usual product construction works. Moreover, if one takes a product $\mathcal{A} \times \mathcal{QA}$ of a tree automaton and a single-run QA (where selecting states are pairs containing a selecting state of $\mathcal{QA}$), the result is a single-run QA, and the nonemptiness problem for it is solvable in polynomial time too. This makes single-run QAs convenient for reasoning tasks, as outlined in the introduction.

3 Conditional XPath queries

We present a first-order complete extension of XPath, called *conditional XPath*, or CXPath [19]. To keep notations in translations manageable, we introduce very minor modifications to the syntax: we use an existential quantifier **E** instead of the usual XPath node test brackets [ ], and we use symbols for the main XPath axes, rather then their verbose analogs.

The *node formulae* α and *path formulae* β of CXPath are given by:

$$\alpha, \alpha' \quad := \quad a \quad | \quad \neg\alpha \quad | \quad \alpha \vee \alpha' \quad | \quad \mathbf{E}\beta$$

$$\beta, \beta' \quad := \quad ?\alpha \quad | \quad \texttt{step} \quad | \quad \texttt{step}^* \quad | \quad (\texttt{step}/?\alpha)^* \quad | \quad \beta/\beta' \quad | \quad \beta \vee \beta'$$

where $\texttt{step} \in \{\prec_{ch}, \prec_{ch}^-, \prec_{ns}, \prec_{ns}^-\}$.

Given a tree $T = (D, \lambda)$, the semantics of a node formula is a set of nodes $[\![\alpha]\!]_T \subseteq D$, and the semantics of a path formula is a binary relation $[\![\beta]\!]_T \subseteq D \times D$ given by the following rules. We use R^* to denote the reflexive-transitive closure of the relation R, and $\pi_1(R)$ to denote its first projection. The *semantics* is formally defined as follow:

$$\begin{array}{ll}
[\![a]\!]_T = \{s \in D \mid \lambda(s) = a\} & [\![?\alpha]\!]_T = \{(s,s) \mid s \in [\![\alpha]\!]_T\} \\
[\![\neg\alpha]\!]_T = D - [\![\alpha]\!]_T & [\![\texttt{step}]\!]_T = \{(s,s') \mid s, s' \in D \text{ and } (s,s') \in \texttt{step}\} \\
[\![\alpha \vee \alpha']\!]_T = [\![\alpha]\!]_T \cup [\![\alpha']\!]_T & [\![\beta \vee \beta']\!]_T = [\![\beta]\!]_T \cup [\![\beta']\!]_T \\
[\![\mathbf{E}\beta]\!]_T = \pi_1([\![\beta]\!]_T) & [\![\texttt{step}^*]\!]_T = [\![\texttt{step}]\!]_T^* \\
 & [\![\beta/\beta']\!]_T = [\![\beta]\!]_T \circ [\![\beta']\!]_T \\
 & [\![(\texttt{step}/?\alpha)^*]\!]_T = [\![(\texttt{step}/?\alpha)]\!]_T^*
\end{array}$$

CXPath node formulae α define unary queries: such a query selects the set $[\![\alpha]\!]_T$ from a tree T. It is known that these capture precisely unary FO queries on trees [19]. Hence, they can be translated into query automata, although the standard translation from FO into automata necessarily involves non-elementary blowup. The following was shown in [15]:

Theorem 1. *Every* CXPath *node formula α can be translated, in exponential time, into an equivalent single-run query automaton (of exponential size).*

The translation went in two steps:

1. First, CXPath node formulae were translated, in single-exponential time, into formulae of a temporal logic $\mathrm{TL}^{\mathrm{tree}}$. Those formulae could be of size exponential in the size of the original CXPath formula.
2. $\mathrm{TL}^{\mathrm{tree}}$ formulae were translated, again in single-exponential time, into exponential-size QAs.

While this appears to give a double-exponential algorithm, a careful analysis of the translations showed that it did not occur, i.e., the state-space of the resulting QA was always single-exponential in the size of the original CXPath formula.

Still, as we explained before, it is undesirable to have such a two-stage translation, as it practically eliminates any chance of adapting algorithmic techniques that have been developed for the LTL-to-automata translation. So now we provide a direct translation for CXPath to automata.

4 CXPath-to-query automata translation

The direct translation from CXPath α queries into single-run QAs (Q, δ, F, Q_s) is done in the following steps:

1. First, we define the set $\mathrm{sub}(\alpha)$ of subformulae of α (analog of the Fischer-Ladner closure).
2. Then we let the set Q be a subset of $2^{\mathrm{sub}(\alpha)}$ satisfying certain properties.
3. Then we define the set of final states.
4. Selection states are those states $q \subseteq \mathrm{sub}(\alpha)$ that contain α.
5. Finally, we define the transition function: this is the most involved part of the translation.

Intuitively the automaton is such that there is exactly one accepting run for each tree which labels each node by the sets of all subformula from $\mathrm{sub}(\alpha)$ it satisfies.

Subformulae of a CXP formula As standard in translations from logic to automata, we push all the negations down the formula in a way that they are all in front of either an atom (a) or a path formula ($\mathbf{E}\beta$), so we assume all formulae are in such a form (we push negations inside subformulae as well).
We define, for a given node formula α, its subformulae $\mathrm{sub}(\alpha)$ as follows:

$$\begin{array}{ll}
\mathrm{sub}(a) = \{a\} & \mathrm{sub}(\alpha \wedge \alpha') = \{\alpha \wedge \alpha'\} \cup \mathrm{sub}(\alpha) \cup \mathrm{sub}(\alpha') \\
\mathrm{sub}(\neg\alpha) = \{\neg\alpha\} \cup \mathrm{sub}(\alpha) & \mathrm{sub}(\alpha \vee \alpha') = \{\alpha \vee \alpha'\} \cup \mathrm{sub}(\alpha) \cup \mathrm{sub}(\alpha') \\
\mathrm{sub}(\mathtt{step}) = \{\mathtt{step}\} & \mathrm{sub}(\beta \vee \beta') = \{\beta \vee \beta'\} \cup \mathrm{sub}(\beta) \cup \mathrm{sub}(\beta') \\
\mathrm{sub}(\mathtt{step}^*) = \{\mathtt{step}^*\} & \mathrm{sub}(\beta/\beta') = \{\beta/\beta'\} \cup \mathrm{sub}(\beta) \cup \mathrm{sub}(\beta') \\
\mathrm{sub}(?\alpha) = \{?\alpha\} \cup \mathrm{sub}(\alpha) & \mathrm{sub}((\mathtt{step}/?\alpha)^*) = \{(\mathtt{step}/?\alpha)^*\} \cup \mathrm{sub}(\alpha) \\
\mathrm{sub}(\mathbf{E}\beta) = \{\mathbf{E}\beta\} \cup \mathrm{sub}(\beta) &
\end{array}$$

In the following, we refer to $\mathtt{step}$, $\mathtt{step}^*$, and $(\mathtt{step}/?\alpha)^*$ as navigational connectives.

States Now fix a node formula α_0. The set of states of the query automaton $\mathcal{QA}_{\alpha_0}$ is $Q \subset 2^{\mathrm{sub}(\alpha_0)}$ where each $q \in Q$ satisfies the following conditions for each formula $\gamma \in \mathrm{sub}(\alpha_0)$:

- If $\gamma = \alpha_1 \vee \alpha_2$, then $\gamma \in q$ iff either $\alpha_1 \in q$ or $\alpha_2 \in q$.
- If $\gamma = \alpha_1 \wedge \alpha_2$, then $\gamma \in q$ iff both $\alpha_1 \in q$ and $\alpha_2 \in q$.
- If $\gamma = \beta_1 \vee \beta_2$, then $\gamma \in q$ iff either $\beta_1 \in q$ or $\beta_2 \in q$.
- If $\gamma = ?\alpha$, then $\gamma \in q$ iff $\alpha \in q$.
- If $\gamma = \mathbf{E}\beta$, then $\gamma \in q$ iff $\beta \in q$.
- If $\gamma = \beta_1/\beta_2$, where β_1 does not contain any navigational connectives, then $\gamma \in q$ iff $\beta_1 \in q$ and $\beta_2 \in q$.
- If γ is a node formula of the form a or $\mathbf{E}\beta$, then $\gamma \in q$ iff $\neg\gamma \notin q$.
- If $\gamma = \beta_2/\beta_1$ where β_2 is $\mathtt{step}^*$ or $(\mathtt{step}/?\alpha)^*$, then $\beta_1 \in q$ implies $\beta_2/\beta_1 \in q$.
- If $\gamma = \mathtt{step}^*$ or $\gamma = (\mathtt{step}/?\alpha)^*$ then $\gamma \in q$.
- Each state q contains exactly one formula of the form a.

Final states The set of final states $F \subset Q$ is the set of states q such that for any $\mathtt{step} \in \{\prec_{ns}, \prec_{ns}^-, \prec_{ch}^-\}$:

- q does not contain any formula of the form $\mathtt{step}$ or $\mathtt{step}/\beta$.
- If q contains a formula of the form β'/β, where β' is of the form $\mathtt{step}^*$ or $(\mathtt{step}/?\alpha)^*$ then q also contains β.

Selecting states The selecting states Q_s are simply the states containing α_0. In general, we denote by Q_u the set of states containing a certain path or node formula u. Thus $Q_s = Q_{\alpha_0}$.

Transition function The transition function δ is defined as follows, for each $q \in Q$ and $a \in \Sigma$:

$$\delta(q, a) = L_{ns} \cap L_{up}(q) \cap L_{down}(q, a)$$

where the languages $L_{down}(q, a)$, $L_{up}(q)$, and L_{ns} are to be defined shortly. Intuitively the language L_{ns} checks horizontal constraints. It is independent from q and a and describes valid sequences of siblings w.r.t. the formula α_0. The language $L_{up}(q)$ is used to check upward conditions. It ensures that every child of a node assigned to q is assigned to a state such that its upward formula are compatible with q. Finally, the language $L_{down}(q, a)$ is used to check local and downward constraints. For each node labeled by a and assigned to a state q it ensures that the labelling of its children is compatible with the downward formula of q. It also checks that a is compatible with q.

The language L_{ns} of horizontal conditions L_{ns} is the set of words from $q_0 q_1 \cdots q_{n-1} q_n$ for $n \in \mathbb{N}$ which respects the following constraints.

- Conditions for the leftmost state q_0
  - q_0 does not contain any formula of the form $\prec_{ns}^- /\beta$
  - From each formula $\beta'/\beta \in \mathrm{sub}(\alpha_0)$ where β' is either $\prec_{ns}^{-*}$ or $(\prec_{ns}^- /?\alpha)^*$, $\beta \in q_0$ iff $\beta'/\beta \in q_0$.

- Conditions for the rightmost state q_n
  - q_n does not contain any formula of the form $\prec_{ns} /\beta$
  - From each formula $\beta'/\beta \in \mathrm{sub}(\alpha_0)$ where β' is either $\prec_{ns}^*$ or $(\prec_{ns} /?\alpha)^*$, $\beta \in q_n$ iff $\beta'/\beta \in q_n$.
- Conditions for two consecutive states q_i, q_{i+1}.
  - For any path formula β in q_i
    - If $\beta = \prec_{ns} /\beta'$ then $\beta' \in q_{i+1}$.
    - If $\beta = \prec_{ns}^* /\beta'$ then either $\beta' \in q_i$ or $\prec_{ns}^* /\beta' \in q_{i+1}$.
    - If $\beta = (\prec_{ns} /?\alpha)^*/\beta'$ then either $\beta' \in q_i$ or $(\prec_{ns} /?\alpha)^*/\beta' \in q_{i+1}$ and $\alpha \in q_{i+1}$.
  - For any path formula β in q_{i+1}
    - If $\beta = \prec_{ns}^- /\beta'$ then $\beta' \in q_i$.
    - If $\beta = \prec_{ns}^{-*} /\beta'$ then either $\beta' \in q_{i+1}$ or $\prec_{ns}^{-*} /\beta' \in q_i$.
    - If $\beta = (\prec_{ns}^- /?\alpha)^*/\beta'$ then either $\beta' \in q_{i+1}$ or $(\prec_{ns}^- /?\alpha)^*/\beta' \in q_i$ and $\alpha \in q_i$.
  - The set $\mathrm{sub}(\alpha_0) - q_i$ does not contains any formula of the form $\prec_{ns}$ or $\prec_{ns}^*$ or $(\prec_{ns} /?\alpha)^*$.
  - For any formula β in $\mathrm{sub}(\alpha_0) - q_i$
    - If $\beta = \prec_{ns} /\beta'$ then $\beta' \notin q_{i+1}$.
    - If $\beta = \prec_{ns}^* /\beta'$ then $\beta' \notin q_i$ and $\prec_{ns}^* /\beta' \notin q_{i+1}$.
    - If $\beta = (\prec_{ns} /?\alpha)^*/\beta'$ then $\beta' \notin q_i$ and either $(\prec_{ns} /?\alpha)^*/\beta' \notin q_{i+1}$ or $\alpha \notin q_{i+1}$.
  - The set $\mathrm{sub}(\alpha_0) - q_{i+1}$ does not contains any formula of the form $\prec_{ns}^-$ or $\prec_{ns}^{-*}$ or $(\prec_{ns}^- /?\alpha)^*$.
  - For any formula β in $\mathrm{sub}(\alpha_0) - q_{i+1}$
    - If $\beta = \prec_{ns}^- /\beta'$ then $\beta' \notin q_i$.
    - If $\beta = \prec_{ns}^{-*} /\beta'$ then $\beta' \notin q_{i+1}$ and $\prec_{ns}^{-*} /\beta' \notin q_i$.
    - If $\beta = (\prec_{ns}^- /?\alpha)^*/\beta'$ then $\beta' \notin q_{i+1}$ and either $(\prec_{ns}^- /?\alpha)^*/\beta' \notin q_i$ or $\alpha \notin q_i$.

It is easy to build a DFA that checks all the conditions and thus recognises L_{ns}.

The language $L_{up}(q)$ of upward conditions We define $L_{up}(q) = Q_{up}(q)^*$, where $Q_{up}(q)$ is the set of states that are legal children of q wrt formulae of the form β or β/β' where β is $\prec_{ch}^-$, $\prec_{ch}^{-*}$ or $(\prec_{ch}^- /?\alpha)^*$.
Formally, a state q' belongs to $Q_{up}(q)$ if it satisfies the following conditions:

- For each path formula β in q'
  - If $\beta = \prec_{ch}^- /\beta'$ then $\beta' \in q$.
  - If $\beta = \prec_{ch}^{-*} /\beta'$ then either $\beta' \in q'$ or $\prec_{ch}^{-*} /\beta' \in q$.
  - If $\beta = (\prec_{ch}^- /?\alpha)^*/\beta'$ then either $\beta' \in q'$ or $(\prec_{ch}^- /?\alpha)^*/\beta' \in q$ and $\alpha \in q$.
- The set $\mathrm{sub}(\alpha_0) - q'$ does not contains any formula of the form $\prec_{ch}^-$ or $\prec_{ch}^{-*}$ or $(\prec_{ch}^- /?\alpha)^*$.
- For each path formula β in $\mathrm{sub}(\alpha_0) - q'$
  - If $\beta = \prec_{ch}^- /\beta'$ then $\beta' \notin q$.
  - If $\beta = \prec_{ch}^{-*} /\beta'$ then $\beta' \notin q'$ and $\prec_{ch}^{-*} /\beta' \notin q$.
  - If $\beta = (\prec_{ch}^- /?\alpha)^*/\beta'$ then $\beta' \notin q'$ and either $(\prec_{ch}^- /?\alpha)^*/\beta' \notin q$ or $\alpha \notin q$.

The language $L_{down}(q, a)$ of local and downward conditions Let $\mathrm{sub}^{down}(\alpha_0)$ be is the set of formulae from $\mathrm{sub}(\alpha_0)$ of the form either β or β/β' where β is $\prec_{ch}$, $\prec_{ch}^*$ or $(\prec_{ch}/?\alpha)^*$.
We define $L_{down}(q, a) = L(a, q, a) \cap \bigcap_{u \in \mathrm{sub}^{down}(\alpha_0)} L(u, q, a)$ where each $L(u, q, a)$ is defined as follows:

- Propositional letters
  - $L(a, q, a) = Q^*$ if $a \in q$ and $\emptyset$ otherwise.
- Path conditions when $u \in q$
  - If $u = \prec_{ch}$ then $L(u, q, a) = Q^+$.
  - If $u = \prec_{ch}^*$ or $u = (\prec_{ch}/?\alpha)^*$ then $L(u, q, a) = Q^*$.
  - If $u = \prec_{ch}/\beta'$ then $L(u, q, a) = Q^* Q_{\beta'} Q^*$.
  - If $u = \prec_{ch}^*/\beta'$ then either $\beta' \in q$ or $L(u, q, a) = Q^* Q_{\prec_{ch}^*/\beta'} Q^*$.
  - If $u = (\prec_{ch}/?\alpha)^*/\beta'$ then either $\beta' \in q$ or $L(u, q, a) = Q^*(Q_{(\prec_{ch}/?\alpha)^*/\beta'} \cap Q_\alpha)Q^*$.
- Path conditions when $u \notin q$
  - If $u = \prec_{ch}$ then $L(u, q, a) = \epsilon$.
  - If $u = \prec_{ch}^*$ or $u = (\prec_{ch}/?\alpha)^*$ then $L(u, q, a) = \emptyset$.
  - If $u = \prec_{ch}/\beta'$ then $L(u, q, a) = \overline{Q^* Q_{\beta'} Q^*}$.
  - If $u = \prec_{ch}^*/\beta'$ then either $\beta' \in q$ and $L(u, q, a) = \emptyset$, or $\beta' \notin q$ and $L(u, q, a) = \overline{Q^* Q_{\prec_{ch}^*/\beta'} Q^*}$.
  - If $u = (\prec_{ch}/?\alpha)^*/\beta'$ then either $\beta' \in q$ and $L(u, q, a) = \emptyset$, or $\beta' \notin q$ and $L(u, q, a) = \overline{Q^*(Q_{(\prec_{ch}/?\alpha)^*/\beta'} \cap Q_\alpha)Q^*}$.

5 Correctness of the construction

In this section, we prove that the correctness of our construction.

Theorem 2. *Let α_0 be a* CXPath *node formula.*

1. *The automaton $\mathcal{QA}_{\alpha_0}$ is a single-run query automaton such that $[\![\alpha_0]\!]_T = \mathcal{QA}_{\alpha_0}(T)$ for every tree T.*

2. *The automaton $\mathcal{QA}_{\alpha_0}$ is of size exponential in the size of α_0 and can be constructed in single-exponential time*

The second point of the theorem follows directly from the construction given in the previous section. The first point of Theorem 2 follows from two propositions, presented below.

Consistency and Maximality We show that an accepting run of the automaton labels each node of the input tree by a state which represents exactly all subformulae of the original formula which are true at the node. As an immediate corollary, the automaton has at most one accepting run per tree, and that those accepting runs select the same nodes as the CXPath formula.

First, we show that each accepting run of the automaton is both consistent and maximal in the following sense. Recall that our trees are of the form $T = (D, \lambda)$, where D is the domain, and λ is the labelling function. Throughout the section, we fix a node CXPath formula α_0.

Definition 1 (Consistency). *Let ρ be a labelling function $D \to 2^{\mathrm{sub}(\alpha_0)}$, for a given tree $T(D, \lambda)$ and a given node formula α_0. We say that ρ is* consistent *if, for each node $s \in D$, the following is true:*

– *for each node formula $\alpha \in \rho(s)$, we have $s \in [\![\alpha]\!]_T$;*
– *for each path formula $\beta \in \rho(s)$, there is at least one pair (s, s') in $[\![\beta]\!]_T$.*

Definition 2 (Maximality). *Let ρ be a labelling function $D \to 2^{\mathrm{sub}(\alpha_0)}$, for a given tree $T(D, \lambda)$ and a given node formula α_0. We say that ρ is* maximal *if, for each node $s \in D$, the following is true:*

– *for each node formula $\alpha \in \mathrm{sub}(\alpha_0)$, if $s \in [\![\alpha]\!]_T$, then $\alpha \in \rho(s)$;*
– *for each path formula $\beta \in \mathrm{sub}(\alpha_0)$, if $[\![\beta]\!]_T$ contains at least one pair of the form (s, s'), then $\beta \in \rho(s)$.*

Proposition 2 (Consistency and maximality). *If ρ is an accepting run of $\mathcal{QA}_{\alpha_0}$ on a given tree T and for a given node formula α_0, then ρ is both consistent and maximal.*

This proposition is proved by a mutual induction on the structure of node and path formulae occurring in $\mathrm{sub}(\alpha_0)$. We present a few sample cases below; full details are in the appendix.

Let s be a node of the tree, a its label, α a node formula, and β a path formula. We suppose, in the cases of consistency, that both α and β are in $\rho(s)$, and in the cases of maximality, that both α and β are true at s. In the following we denote by *basic constraints*, the constraints given in the definition of the states of the automaton in Section 4.

– **Consistency**: some base cases.
  - If $\alpha = a'$ then $a' = a$ so α is true at s.(Otherwise, $a \notin \rho(s)$ as each states contains at most one propositional letter so $L(a, \rho(s), a)$ is empty, and thus ρ cannot be an accepting run.)
  - If $\beta = \prec_{ch}$ then $L(\prec_{ch}, \rho(s), a) = Q^+$, thus s has a least one child, so β is true at s.
  - If $\beta = \prec_{ch}^-$ then the only case where β might not be true is at the root. But this cannot happen, as any state containing $\prec_{ch}^-$ cannot be final. So β is true at s.

– **Maximality**: some base cases.
  - If $\alpha = a'$ then $a' = a$, because α is true at s. If $\alpha \notin \rho(s)$ then $L(a, \rho(s), a) = \emptyset$ and ρ cannot be accepting. Thus, $\alpha \in \rho(s)$.
  - If $\beta = \prec_{ch}$ then s is not a leaf. If $\beta \notin \rho(s)$ then $L(\beta, \rho(s), a) = \epsilon$ which contradicts the fact that s is not a leaf. Thus, $\beta \in \rho(s)$.
  - If $\beta = \prec_{ch}^-$ then s is not the root, which implies that s has a parent s', labelled by the state $\rho(s')$. Then, we have $\rho(s) \in L_{up}(\rho(s'))$, and all states in L_{up} contain β. Thus, $\beta \in \rho(s)$.

– **Consistency**: some induction cases.
  - If $\alpha = \neg\alpha'$ then, because of basic constraints, $\alpha' \notin \rho(s)$. By the induction hypothesis for maximality, α' cannot be true at s. So α is true at s.

  - If $\alpha = \alpha_1 \vee \alpha_2$ then, because of basic constraints, either α_1 or α_2 is in $\rho(s)$. Then, by the induction hypothesis for consistency, either α_1 or α_2 is true at s. So α is true at s.
  - If $\alpha = \mathbf{E}\beta$ then, because of basic constraints, $\beta \in \rho(s)$. By the induction hypothesis for consistency, β is true at s, i.e. contains at least one element when evaluated starting from s. So α is true at s.
  - If $\beta = ?\alpha$ then, because of basic constraints, $?\alpha \in \rho(s)$. By the induction hypothesis for consistency, α is true at s. So the interpretation of β contains at least (s, s), hence β is satisfied at s.
- **Maximality**: some induction cases.
  - If $\alpha = \neg\alpha'$ then, α' cannot be true at s. By the induction hypothesis for consistency, this means that $\alpha' \notin \rho(s)$. Then, because of basic constraints, $\alpha \in \rho(s)$.
  - If $\alpha = \alpha_1 \vee \alpha_2$ then either α_1 or α_2 is true at s. By the induction hypothesis for maximality, either α_1 or α_2 is in $\rho(s)$. So, because of basic constraints, $\alpha \in \rho(s)$.
  - If $\alpha = \mathbf{E}\beta$ then the interpretation of β contains (s, s') for some s', which means that β is true at s. By the induction hypothesis, $\beta \in \rho(s)$. So, because of basic constraints, $\alpha \in \rho(s)$.
  - If $\beta = ?\alpha$ then the interpretation of β contains (s, s) which means that α is true at s. By the induction hypothesis, $\alpha \in \rho(s)$. So, because of basic constraints, $\beta \in \rho(s)$.

Accepting Runs We have proved so far that the accepting runs of the automaton were consistent and maximal labellings, which means that the automaton accepts at most the same trees than the original CXPath formula. We now prove that each consistent and maximal labelling of a tree is an accepting run of the automaton.

Proposition 3 (Accepting Runs). *Let ρ be a maximal and consistent labelling of the nodes of a given tree T with respect to a given formula α_0. Then ρ is an accepting run of $\mathcal{QA}_{\alpha_0}$.*

This shows that the QA we constructed has indeed at most one accepting run per tree, and that during this run, a node gets labelled by a state which contains α_0 iff α_0 is true at that node. Hence, the automaton selects all the nodes which are also selected by α_0, and this concludes the proof of Theorem 2.

Acknowledgements Most of this work was done while all three authors were at the University of Edinburgh, supported by EPSRC grant G049165 and FET-Open Project FoX, grant agreement 233599.

References

1. S. Abiteboul, B. Cautis, T. Milo. Reasoning about XML update constraints. In *PODS'07*, pages 195–204.
2. M. Arenas, W. Fan, L. Libkin. Consistency of XML specifications. In *Inconsistency Tolerance*, Springer, 2005, pages 15–41.

3. M. Benedikt, W. Fan, F. Geerts. XPath satisfiability in the presence of DTDs. In *PODS'05*, pages 25–36.
4. M. Bojanczyk, C. David, A. Muscholl, Th. Schwentick, L. Segoufin. Two-variable logic on data trees and XML reasoning. In *PODS'06*, pages 10–19.
5. D. Calvanese, G. De Giacomo, M. Lenzerini, M. Y. Vardi. Regular XPath: constraints, query containment and view-based answering for XML documents. In *Logic in Databases*, 2008.
6. E. Clarke, O. Grumberg, D. Peled. *Model Checking*, MIT Press, 1999.
7. R. Clark. Querying streaming XML using visibly pushdown automata. Technical Report, University of Illinois, 2008.
8. M. Daniele, F. Giunchiglia, M.Y. Vardi. Improved automata generation for linear temporal logic. In *CAV'99*, pages 249–260.
9. M. Frick, M. Grohe, C. Koch. Query evaluation on compressed trees. In *LICS'03*, pages 188-197.
10. P. Genevés and N. Layaida. A system for the static analysis of XPath. *ACM TOIS* 24, 2006, 475–502.
11. P. Genevés, N. Layaida, and A. Schmitt. Efficient static analysis of XML paths and types. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation*, 2007, pages 342–351.
12. R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV 1995*, pages 3–18.
13. A. K. Gupta, D. Suciu Stream processing of XPath queries with predicates. In *SIGMOD 2003*, pages 419–430.
14. L. Libkin. Logics for unranked trees: an overview. In *ICALP'05*, pages 35-50.
15. L. Libkin, C. Sirangelo. Reasoning about XML with temporal logics and automata. *Journal of Applied Logic* 8 (2010), 210–232.
16. S. Maneth, T. Perst, H. Seidl. Exact XML type checking in polynomial time. In *ICDT 2007*, pages 254–268.
17. W. Martens, F. Neven, Th. Schwentick. Simple off the shelf abstractions for XML schema. *SIGMOD Record* 36(3): 15–22 (2007).
18. M. Marx. XPath with conditional axis relations. In *EDBT 2004*, pages 477–494.
19. M. Marx. Conditional XPath. *ACM TODS* 30 (2005), 929–959.
20. F. Neven. *Design and Analysis of Query Languages for Structured Documents.* PhD Thesis, U. Limburg, 1999.
21. F. Neven. Automata, logic, and XML. In *CSL 2002*, pages 2–26.
22. F. Neven, Th. Schwentick. Query automata over finite trees. *TCS*, 275 (2002), 633–674.
23. F. Neven, J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. *J. ACM* 49(1): 56–100 (2002).
24. J. Niehren, L. Planque, J.-M. Talbot, S. Tison. N-ary queries by tree automata. In *DBPL 2005*, pages 217–231.
25. B.-H. Schlingloff. Expressive completeness of temporal logic of trees. *Journal of Applied Non-Classical Logics* 2 (1992), 157–180.
26. Th. Schwentick. XPath query containment. *SIGMOD Record* 33 (2004), 101–109.
27. Th. Schwentick. Automata for XML – a survey. *JCSS* 73 (2007), 289–315.
28. W. Thomas. Languages, automata, and logic. In Handbook of Formal Languages, volume III, pages 389–455.
29. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. Banff Higher Order Workshop, 1996.
30. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf.& Comput.* 115 (1994), 1–37.