The Complexity of Query Containment in Expressive Fragments of XPath 2.0

BALDER TEN CATE

INRIA and ENS Cachan, France

AND

CARSTEN LUTZ

Universität Bremen, Germany

Abstract. XPath is a prominent W3C standard for navigating XML documents that has stimulated a lot of research into query answering and static analysis. In particular, query containment has been studied extensively for fragments of the 1.0 version of this standard, whereas little is known about query containment in (fragments of) the richer language XPath 2.0. In this article, we consider extensions of CoreXPath, the navigational core of XPath 1.0, with operators that are part of or inspired by XPath 2.0: path intersection, path equality, path complementation, for-loops, and transitive closure. For each combination of these operators, we determine the complexity of query containment, both with and without DTDs. It turns out to range from EXPTIME (for extensions with path equality) and 2-EXPTIME (for extensions with path intersection) to non-elementary (for extensions with path complementation or for-loops). In almost all cases, adding transitive closure on top has no further impact on the complexity. We also investigate the effect of dropping the upward and/or sibling axes, and show that this sometimes leads to a reduction in complexity. Since the languages we study include negation and conjunction in filters, our complexity results can equivalently be stated in terms of satisfiability. We also analyze the above languages in terms of succinctness.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Languages—*Query languages*

General Terms: Languages, Algorithms

Additional Key Words and Phrases: XML, XPath, containment, satisfiability, complexity

---

An extended abstract of this article appeared in *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2007)*, ACM, New York, pp. 73–82.

B. ten Cate was supported by NWO research grant 639.021.508. C. Lutz was supported by the EU-funded IST-2005-7603 FET Project Thinking Ontologies (TONES).

Authors' addresses: B. ten Cate, ENS Cachan, 61, avenue du President Wilson, 94235 CACHAN Cedex, France, e-mail: balder.tencate@uva.nl; C. Lutz, Universität Bremen, Fachbereich 03, Postfach 330440, ENS Cachan 28334 Bremen, Germany, e-mail: clu@informatik.uni-bremen.de.

DOI 10.1145/1568318.1568321 http://doi.acm.org/10.1145/1568318.1568321

ACM Reference Format:
ten Cate, B., and Lutz, C. 2009. The complexity of query containment in expressive fragments of XPath 2.0. J. ACM 56, 6, Article 31 (September 2009), 48 pages.
DOI = 10.1145/1568318.1568321 http://doi.acm.org/10.1145/1568318.1568321

1. *Introduction*

The growing popularity of XML as a standard for storage and exchange of semi-structured data has led to the introduction of a large number of XML-related formalisms, most notably schema languages such as DTDs and XML Schema, and query and transformation languages such as XQuery and XSLT. Located at the heart of most of these is XPath, the basic formalism for navigating through XML documents. For example, XPath is used in XML Schema to define keys [Gao et al. 2007], in XQuery to bind variables [Boag et al. 2007], and in XSLT to select nodes in XML documents [Kay 2007]. Like XML, XPath is a standard of the W3C committee. Since XPath 1.0 was first released in November 1999 [Clark and DeRose 1999], the development of other XML-related standards such as XQuery has suggested many changes. Consequently, the first working draft for a new version of XPath was published in 2001, but it was not until January 2007 that the official W3C recommendation for XPath 2.0 was released [Berglund et al. 2007], together with the one for XQuery 1.0.

Because of the central role that XPath plays in many XML-related formalisms, the static analysis of XPath expressions is a prominent subject in research about XML processing. In particular, *containment* and *satisfiability* for XPath expressions are of prime importance. Due to the presence of data value comparisons in the XPath standard, both problems are undecidable for full XPath 1.0 [Benedikt et al. 2008]. However, the full expressive power of XPath is rarely used, and consequently there is a large body of work about the decidability and complexity of static analysis problems for *fragments* of XPath 1.0, often in the presence of document schemas (DTDs) and other constraints such as functional dependencies [Miklau and Suciu 2004; Deutsch and Tannen 2005; Arenas et al. 2008; Neven and Schwentick 2006; Benedikt et al. 2008]. The emerging picture indicates that it is useful to distinguish between those fragments of XPath that admit negation in filter expressions, called positive fragments, and those that do not. Regarding positive fragments of XPath 1.0, there are some simple cases for which containment and satisfiability are tractable, but even modest extensions push the complexity to (co)NP or PSPACE. [Schwentick 2004]. If negation in filter expressions is admitted, the complexity is usually even higher: EXPTIME is typical, but some more difficult cases have also been identified [Benedikt et al. 2008].

In contrast to XPath 1.0, static analysis for XPath 2.0 has not been extensively studied. A notable exception is Hidders [2003], which is concerned with satisfiability for positive fragments of XPath 2.0. The situation turns out to be similar to containment in XPath 1.0, that is, the complexity is between polytime and PSPACE. The aim of the current paper is *to analyze the complexity of static analysis in fragments of XPath 2.0 that admit negation of node expressions*. Our analysis is in terms of containment, but applies to satisfiability as well since containment and (un-)satisfiability are polynomially inter-reducible in the XPath dialects considered here. We analyze the case with and without (extended) DTDs, give tight complexity

bounds for a number of relevant cases, and also present some observations concerning succinctness.

Our Contributions. We extend CoreXPath, the navigational core of XPath 1.0 identified in Gottlob and Koch [2002] and Gottlob et al. [2005], with the following ingredients of XPath 2.0: *path intersection* ($\cap$), *path complementation* ($-$), and *iteration* (for). Besides these three operators, we also consider *(reflexive-)transitive closure* ($*$) and *path equalities* ($\approx$). Path equalities, also known as *node set equalities* and not to be confused with data value equalities, are not part of XPath 2.0 as a primitive construct, but can be expressed. They have been studied in Deutsch and Tannen [2005], Benedikt et al. [2005], Olteanu et al. [2002], and ten Cate [2006]. Transitive closure is not part of XPath 2.0 and cannot be expressed, but it extends the expressive power of XPath in a very natural way. For instance, it allows us to express DTDs and other schema constraints inside XPath [Marx 2004], and it enables view based query rewriting for recursive XML views [Fan et al. 2007].

These five additions to CoreXPath are not all independent: path equalities can be expressed using path intersection, which can in turn be expressed using path complementation, which can again be expressed using iteration. Thus, we obtain a hierarchy regarding expressive power, which is depicted in Figure 1. It is based on expressivity results from Marx [2005], Marx and de Rijke [2005], and ten Cate [2006]. The figure also lists equivalent variants of first order logic, where $UCQ[FO^2]$ refers to unions of conjunctive queries, where we allow arbitrary FO^2-definable unary queries as atomic predicates. $FO{+}TC^1$ refers to first order logic with monadic transitive closure, and FO^* refers to first-order logic with nonparametrized monadic transitive closure, cf. [ten Cate 2006].

For each of the languages shown in Figure 1, we determine the complexity of containment for path expressions, showing that it ranges from ExpTime to non-elementary. Our main results are summarized in Table 1; because of the mentioned inter-reducibility, they apply to both containment and satisfiability. We prove most upper bounds in the absence of DTDs since DTDs can be expressed in CoreXPath($*$) with only a linear blowup in size. The same holds even for *extended DTDs* (EDTDs), an extension of the DTD formalism first introduced in Papakonstantinou and Vianu [2000] under the somewhat misleading name of "specialized DTDs". Notably, EDTDs capture all regular tree languages. Only for the downward fragment of CoreXPath($\cap$), we prove the upper bound directly with reference to EDTDs. Since containment without DTDs can be reduced to containment with DTDs, the lower bounds carry over as well. Thus, the results in Table 1 apply to containment and satisfiability without DTDs, with DTDs, and with EDTDs.

Our results show that $\approx$ and $*$ never increase the complexity of the containment problem, with one exception: adding $*$ to the downward fragment of CoreXPath($\cap$) increases the complexity from ExpSpace to 2-ExpTime. Adding $\cap$ usually increases the complexity of containment by one exponential, even though $\cap$ does not give more expressive power than $\approx$. Finally, the effect of adding path complementation or for is rather devastating, as each of these operators renders containment hard for non-elementary time. In other words, the amenities of XPath 2.0 come at the price of an increase in computational complexity, at least in the presence of negation. We can also conclude that CoreXPath($*$, $\approx$) is a rather well behaved fragment. Among all languages studied in this paper, it is the most expressive one for which containment is still decidable in ExpTime and thus not more difficult than in CoreXPath, and

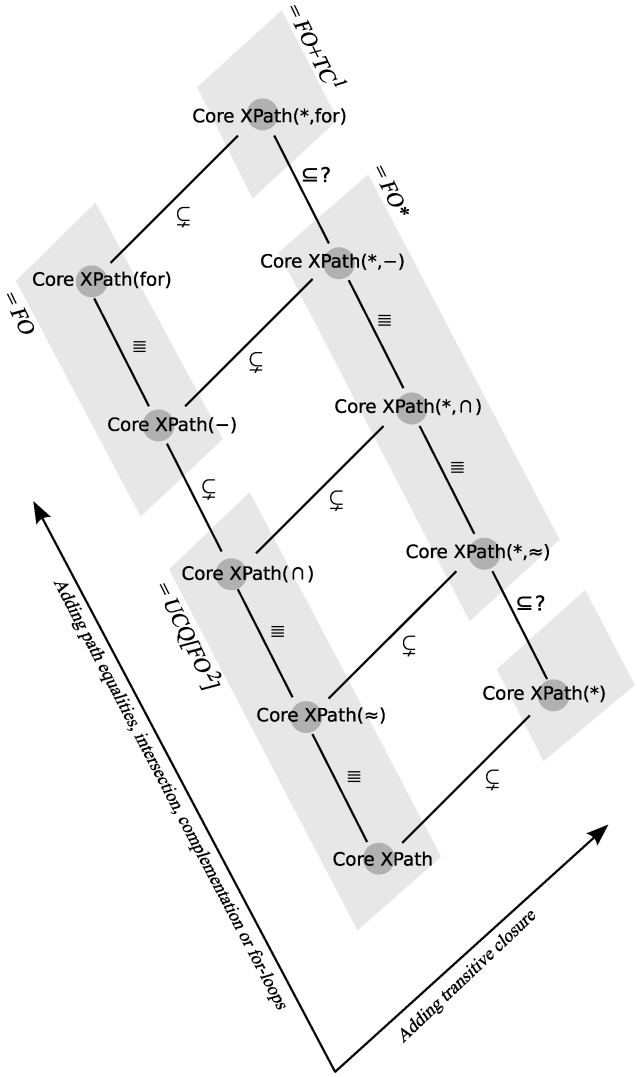


FIG. 1. Hierarchy of XPath languages.

this holds even in the presence of DTDs. Note that even for the positive downward fragment of CoreXPath, containment in the presence of DTDs is already EXPTIME-hard [Neven and Schwentick 2006].

Our observations concerning succinctness are as follows. Consider each class of equally expressive languages in Figure 1, starting with the languages that are expressively complete for $UCQ[FO^2]$. We show that CoreXPath($\cap$) is exactly exponentially more succinct than CoreXPath and CoreXPath($\approx$), explaining the higher computational complexity of the former. Whether CoreXPath($\approx$) is exponentially more succinct than CoreXPath is an open question. Moving on to the languages that are expressively complete for FO^*, we show that CoreXPath($*$, $\cap$) is at least exponentially and at most doubly exponentially more succinct than CoreXPath($*$, $\approx$), and CoreXPath($*$, $-$) is nonelementarily more succinct than CoreXPath($*$, $\cap$).

TABLE I. SUMMARY OF OUR COMPLEXITY RESULTS

	CoreXPath(...)	CoreXPath(∗, ...)
for	Non-elementary (even for the one-variable downward fragment of CoreXPath(for))	
−	Non-elementary (even for the downward fragment of CoreXPath(−))	
∩	2-EXPTIME-complete (hardness holds even for the vertical and forward fragments); EXPTIME-complete when the nesting depth of intersection is bounded EXPSPACE-complete for the downward fragment;	2-EXPTIME-complete (hardness holds even for the downward fragment); EXPTIME-complete when the nesting depth of intersection is bounded
≈	EXPTIME-complete (hardness holds even for the downward fragment of CoreXPath)	

All results apply to containment and satisfiability of XPath expressions, also in the presence of (E)DTDs.

Finally, concerning languages that are expressively complete for *FO*, we show that CoreXPath(for) is at least exponentially more succinct than CoreXPath(−).

Related Work. Since there is an overwhelming number of complexity results for the static analysis of XPath fragments, we confine ourselves to a rough overview of the literature and refer the reader to Schwentick [2004] for a recommendable, though nowadays somewhat outdated survey. As before, we distinguish between positive fragments and fragments that include negation in filter expressions.

For positive fragments, containment and satisfiability often have different complexity. Classical papers on the complexity of query containment include Miklau and Suciu [2004], Wood [2003], and Neven and Schwentick [2006], which all analyze the border of tractability, with and without DTDs. Deutsch and Tannen [2005] additionally take into account integrity constraints and identify fragments of different complexity, ranging from polytime to undecidable. Satisfiability in positive fragments with DTDs is studied in Benedikt et al. [2008] without sibling axes, and in Geerts and Fan [2005] with sibling axes. Both papers also address fragments with negation. In Arenas et al. [2008] and Lakshmanan et al. [2004], the focus is on satisfiability in positive fragments in the presence of both DTDs and constraints. All the mentioned papers consider XPath 1.0. As already mentioned, Hidders studies satisfiability in positive fragments of XPath 2.0 [Hidders 2003].

The fundamental complexity result for fragments with negation is that containment and satisfiability in CoreXPath are EXPTIME-complete, even when transitive closure is added [Marx 2004]. There appear to be only a few known fragments with higher complexity. For example, Benedikt et al. [2008] identifies fragments that include data value equalities and are NEXPTIME-complete or even undecidable. Some first evidence that adding intersection to CoreXPath is likely to increase computational complexity can be derived from the relationship between XPath and Propositional Dynamic Logic (PDL), and the 2EXPTIME-hardness result for PDL with intersection in Lange and Lutz [2005], as well as from the EXPSPACE-hardness of equivalence for semi-extended regular expressions [Fürer 1980].

Other articles related to our work include the following. Benedikt et al. [2005] studies the closure under intersection and complementation of various fragments of XPath 1.0. Tajima and Fukui [2004], Böttcher [2004], and Hammerschmidt et al. [2005] study XPath with intersection and (sometimes) complementation, motivated by different applications such as redundancy elimination in answers to multiple XPath queries, node selection in queries to distributed XML documents, and index updating. Finally, ten Cate [2006] studies the expressive power of XPath with transitive closure (and node set equalities), and [ten Cate and Marx 2009] gives an axiomatization of XPath 2.0.

2. *Preliminaries*

We review the syntax and semantics of CoreXPath and its relevant extensions. We also introduce and compare various static analysis problems.

2.1. XML Trees and EDTDs.

For the purposes of this article, an XML document is a finite node-labeled sibling-ordered tree. The node labels represent XML tags. As usual, we abstract away from all data associated with the nodes of an XML document other than XML tags. We use the term *XML tree* for this type of structures. Throughout this article, we fix a countably infinite set Σ of labels.

Definition 1 (*XML Trees*). An *XML tree* is a structure $T = (N, R_\downarrow, R_\rightarrow, L)$, where $(N, R_\downarrow)$ is a finite, rooted tree with child relation $R_\downarrow$, $R_\rightarrow$ is a successor relation on (ordered) siblings, and $L : N \to \Sigma$ assigns a label to each node. $R_\uparrow$ and $R_\leftarrow$ denote the converse of the relations $R_\downarrow$ and $R_\rightarrow$.

In many applications, it is useful to restrict attention to XML trees satisfying a given schema. There is a wealth of schema languages for XML (see, e.g., Thompson et al. [2004], Klarlund et al. [2002], Clark and Murata [2001], and Murata et al. [2005], Martens et al. [2006]). Since the focus of this article is not on such languages, we only consider *extended document type definitions (EDTDs)* as a typical example. EDTDs have first been introduced (under the name specialized DTDs) in Papakonstantinou and Vianu [2000]. As in Papakonstantinou and Vianu [2000], we abstract away from features such as default values and attributes.

Definition 2 (*EDTDs*). An EDTD is a tuple (Δ, P, r, μ), where Δ is a finite set of abstract labels, P is a function that assigns to each element of Δ a regular expression over Δ, $r \in \Delta$ is the *root type*, and μ is a mapping from Δ to Σ.

An XML tree $T = (N, R_\downarrow, R_\rightarrow, L)$ *conforms to* an EDTD $D = (\Delta, P, r, \mu)$ if there is a function $L' : N \to \Delta$ such that

(i) the root of T is mapped to r,
(ii) for each node n with (ordered) children $n_1, \ldots, n_k$, the word $L'(n_1), \ldots, L'(n_k)$ belongs to the language generated by $P(L'(n))$, and
(iii) for all $n \in N$, $L(n) = \mu(L'(n))$.

Standard (nonextended) DTDs correspond to those EDTDs (Δ, P, r, μ) for which $\Delta = \Sigma$ and μ is the identity map. A simple example of an EDTD that cannot be written as a DTD is (Δ, P, r, μ), where $\Delta = \{s_1, s_2, s_3\}$, $P(s_1) = s_2 + \varepsilon$, $P(s_2) = s_3 + \varepsilon$, $P(s_3) = \varepsilon$, $r = s_1$, and $\mu(s_i) = s$ for all $i \leq 3$. Intuitively, we may think of the label s as defining sections in texts. The XML trees conforming to the given EDTD, then, correspond to texts where the nesting depth of sections

TABLE II. SEMANTICS OF CoreXPath

$[\![\tau]\!]_{\mathsf{PExpr}}$	$=$	R_τ
$[\![\tau^*]\!]_{\mathsf{PExpr}}$	$=$	the reflexive transitive closure of $[\![\tau]\!]_{\mathsf{PExpr}}$
$[\![.]\!]_{\mathsf{PExpr}}$	$=$	$\{(n,n) \mid n \in N\}$
$[\![\alpha/\beta]\!]_{\mathsf{PExpr}}$	$=$	$\{(n,m) \mid \exists k.(n,k) \in [\![\alpha]\!]_{\mathsf{PExpr}}$ and $(k,m) \in [\![\beta]\!]_{\mathsf{PExpr}}\}$
$[\![\alpha \cup \beta]\!]_{\mathsf{PExpr}}$	$=$	$[\![\alpha]\!]_{\mathsf{PExpr}} \cup [\![\beta]\!]_{\mathsf{PExpr}}$
$[\![\alpha[\varphi]\,]\!]_{\mathsf{PExpr}}$	$=$	$\{(n,m) \in [\![\alpha]\!]_{\mathsf{PExpr}} \mid m \in [\![\varphi]\!]_{\mathsf{NExpr}}\}$
$[\![p]\!]_{\mathsf{NExpr}}$	$=$	$\{n \in N \mid L(n) = p\}$
$[\![\langle\alpha\rangle]\!]_{\mathsf{NExpr}}$	$=$	$\{n \in N \mid \exists m \in N.\,(n,m) \in [\![\alpha]\!]_{\mathsf{PExpr}}\}$
$[\![\top]\!]_{\mathsf{NExpr}}$	$=$	N
$[\![\neg\varphi]\!]_{\mathsf{NExpr}}$	$=$	$N \setminus [\![\varphi]\!]_{\mathsf{NExpr}}$
$[\![\varphi \wedge \psi]\!]_{\mathsf{NExpr}}$	$=$	$[\![\varphi]\!]_{\mathsf{NExpr}} \cap [\![\psi]\!]_{\mathsf{NExpr}}$

is at most 3. This class of XML trees is not definable by a standard DTD, as follows directly from the characterization of the expressive power of DTDs given in Papakonstantinou and Vianu [2000].

2.2. CoreXPath AND EXTENSIONS. The CoreXPath language was introduced in Gottlob and Koch [2002] and Gottlob et al. [2005] in order to capture the navigational core of XPath 1.0. Here, we follow the definition of CoreXPath given in Marx [2005], which is slightly more expressive. We comment on the difference below.

CoreXPath is a two-sorted language. The primary expressions are *path expressions* which define binary relations on the nodes of an XML tree. Inside these path expressions, *node expressions* may occur, which define sets of nodes. The two types of expressions are defined by simultaneous induction.

Definition 3 (CoreXPath). The node expressions (denoted $\alpha, \beta, \ldots$) and path expressions (denoted $\varphi, \psi, \ldots$) of CoreXPath are defined by simultaneous induction, as follows:

$$\text{Path expressions}: \quad \alpha ::= \tau \mid \tau^* \mid . \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\varphi] \qquad (\tau \in \{\downarrow, \uparrow, \rightarrow, \leftarrow\})$$
$$\text{Node expressions}: \quad \varphi ::= p \mid \langle\alpha\rangle \mid \top \mid \neg\varphi \mid \varphi \wedge \psi \qquad (p \in \Sigma)$$

The semantics of CoreXPath relative to an XML tree $T = (N, R_\downarrow, R_\rightarrow, L)$ is given by two functions $[\![\cdot]\!]^T_{\mathsf{PExpr}}$ and $[\![\cdot]\!]^T_{\mathsf{NExpr}}$. The former maps path expressions to binary relations on N, and the latter node expressions to subsets of N. These functions are defined in Table II, where the superscript $\cdot^T$ is omitted for readability.

An example of a CoreXPath path expression in our notation is $\downarrow^+[p \wedge \neg\langle\downarrow[q]\rangle]$, which selects all descendants with label p that do not have a child with label q. In the official XPath notation, this expression is written as `descendant::p[not(child::q)]` or, using the abbreviated syntax, `.//p[not(q)]`. Notice that an expression such as `child::q` can be both a path expression and a node expression in the official syntax, whereas our syntax disambiguates the two readings by requiring angled brackets $\langle\cdot\rangle$ whenever a path expressions is used as a node expression.

Atomic path expressions of the form τ, τ^* and ".", with $\tau \in \{\downarrow, \uparrow, \rightarrow, \leftarrow\}$, are called *axes*, and in path expressions of the form $\alpha[\varphi]$, φ is called a *filter expression*. The XPath 1.0 and 2.0 specifications [Clark and DeRose 1999; Berglund et al. 2007], as well as the definition of CoreXPath in Gottlob et al. [2005], lack the sibling axes

$\leftarrow$ and $\rightarrow$ and their reflexive-transitive closures $\leftarrow^*$ and $\rightarrow^*$, and include only the irreflexive-transitive closures $\leftarrow^+$ and $\rightarrow^+$ (under the names following-sibling and preceding-sibling). In XPath 1.0 and 2.0, $\leftarrow$, $\rightarrow$, $\leftarrow^*$ and $\rightarrow^*$ are definable from $\leftarrow^+$ and $\rightarrow^+$ using positional predicates and union, but since positional predicates are not included in CoreXPath, $\leftarrow$ and $\rightarrow$ are in fact not definable in CoreXPath as defined in Gottlob et al. [2005]. Following Marx [2004], we have chosen to include $\leftarrow$, $\rightarrow$, $\leftarrow^*$ and $\rightarrow^*$ as the horizontal axes, while $\leftarrow^+$ and $\rightarrow^+$ are definable as $\leftarrow/\leftarrow^*$ and $\rightarrow/\rightarrow^*$, respectively. However, all our results hold independently of this choice. In particular, in lower bound arguments we only make use of the irreflexive-transitive sibling axes $\leftarrow^+$ and $\rightarrow^+$.

Arguably, the most important features of XPath that are *not* included in CoreXPath are attributes and data value comparisons. It is known that, already for the extension of CoreXPath with data value equalities of the form $\alpha/@a = \beta/@b$ and $\alpha/@a =$ 'c', containment is undecidable [Benedikt et al. 2008].

We use τ^+ as shorthand for τ/τ^*, that is, the (nonreflexive) transitive closure of τ. Also, we use $\varphi \Rightarrow \psi$ as a shorthand for the node expression $\neg(\varphi \wedge \neg\psi)$, $\varphi \vee \psi$ as a shorthand for $\neg(\neg\varphi \wedge \neg\psi)$ and $\bot$ as a shorthand for $\neg\top$. Finally, we use $\mathsf{every}(\alpha, \varphi)$ as a shorthand for the node expression $\neg\langle\alpha[\neg\varphi]\rangle$, which expresses that "every node reachable from the current node by α satisfies φ".

We now present five different ways in which the basic CoreXPath language can be extended.

—*Path equalities ($\approx$)* are node expressions of the form $\alpha \approx \beta$. They are interpreted *existentially*:

$$[\![\alpha \approx \beta]\!]_{\mathsf{NExpr}} = \{n \in N \mid \exists m \in N.(n, m) \in [\![\alpha]\!]_{\mathsf{PExpr}} \cap [\![\beta]\!]_{\mathsf{PExpr}}\}.$$

—*Path intersection ($\cap$)* enables path expressions of the form $\alpha \cap \beta$, interpreted as follows:

$$[\![\alpha \cap \beta]\!]_{\mathsf{PExpr}} = [\![\alpha]\!]_{\mathsf{PExpr}} \cap [\![\beta]\!]_{\mathsf{PExpr}}.$$

Since $\alpha \approx \beta$ is equivalent to $\langle\alpha \cap \beta\rangle$, path equalities can be seen as a special case of path intersection.

—*Path complementation ($-$)* enables path expressions of the form $\alpha - \beta$, interpreted as follows:

$$[\![\alpha - \beta]\!]_{\mathsf{PExpr}} = [\![\alpha]\!]_{\mathsf{PExpr}} \setminus [\![\beta]\!]_{\mathsf{PExpr}}.$$

Path intersection can be defined in terms of path complementation: $\alpha \cap \beta$ is equivalent to $U - ((U - \alpha) \cup (U - \beta))$, where U is shorthand for the path expression $\uparrow^*/\downarrow^*$, which defines the universal relation.

—*Iteration (*for*)* enables path expressions of the form "for $\$i$ in α return β", where $\$i$ is from a countably infinite set of *node variables*. In β, it is possible to test equality of the current node with the node bound to $\$i$ using the node expression ". is $\$i$". For example, the path expression "for $\$i$ in α return β[. is $\$i$]" is equivalent to $\alpha \cap \beta$. The precise syntax and semantics will be given in Section 7, where we also show that path complementation can be expressed using iteration.

—*Transitive closure ($*$)* enables path expressions of the form α^*, where α can be any path expression instead of only $\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$ as in basic CoreXPath. It defines

the transitive and reflexive closure of the binary relation defined by α:

$$[\![\alpha^*]\!]_{\mathsf{PExpr}} = \{(n, m) \mid \text{there are } n_1, \ldots, n_k (k \geq 1) \text{ such that } n_1 = n, n_k = m, \\ and (n_i, n_{i+1}) \in [\![\alpha]\!]_{\mathsf{PExpr}} \text{ for } 1 \leq i < k\}$$

Note that, although α^* defines the reflexive-transitive closure of the relation defined by α, for the sake of brevity we refer to $*$ simply as the transitive closure operator.

For any $X \subseteq \{\approx, \cap, -, \mathsf{for}, *\}$, we denote by $\mathsf{CoreXPath}(X)$ the extension of CoreXPath with the operators in X. For any $Y \subseteq \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$, we use $\mathsf{CoreXPath}_Y(X)$ to denote the fragment of $\mathsf{CoreXPath}(X)$ restricted to the following axes: ".", the axes in Y, and the transitive closures of the axes in Y. In particular, we refer to $\mathsf{CoreXPath}_{\downarrow}(X)$ as the *downward* fragment of $\mathsf{CoreXPath}(X)$, to $\mathsf{CoreXPath}_{\downarrow\uparrow}(X)$ as the *vertical* fragment and to $\mathsf{CoreXPath}_{\downarrow\rightarrow}(X)$ as the *forward* fragment.

Examples. To give some examples for the extended versions of CoreXPath, assume that we are working with XML trees that describe books and conform to the EDTD $D = (\Delta, P, \mathsf{Book}, \mu)$, where:

—$\Delta = \{\mathsf{Book}, \mathsf{Chapter}, \mathsf{Section}, \mathsf{Paragraph}, \mathsf{Image}\}$;
—$P(\mathsf{Book}) = \mathsf{Chapter}^+$, $P(\mathsf{Chapter}) = \mathsf{Section}^+$, $P(\mathsf{Section}) = (\mathsf{Section} + \mathsf{Paragraph} + \mathsf{Image})^+$, $P(\mathsf{Paragraph}) = P(\mathsf{Image}) = \varepsilon$;
—$\mu(p) = p$ for all $p \in \Delta$.

Observe that sections can be nested up to any depth. The content of the book is represented by the leafs, and the standard XML document order corresponds to the order of the content in the book. Let following and preceding stand for the CoreXPath path expressions $(\uparrow^*/\rightarrow^+/\downarrow^*)$ and $(\uparrow^*/\leftarrow^+/\downarrow^*)$. Then, the following $\mathsf{CoreXPath}(\approx)$ path expression, evaluated at the root, retrieves the first image of each chapter:

$$\downarrow^*[\mathsf{Image} \wedge \neg(\mathsf{preceding}[\mathsf{Image}] \approx \uparrow^+[\mathsf{Chapter}]/\downarrow^+[\mathsf{Image}])]$$

The following $\mathsf{CoreXPath}(\cap)$ path expression, evaluated at a node, retrieves all following images in the same chapter:

$$\mathsf{following}[\mathsf{Image}] \cap (\uparrow^+[\mathsf{Chapter}]/\downarrow^+[\mathsf{Image}])$$

Next, suppose one wants to retrieve only the first following image in the same chapter. This can naturally be expressed in $\mathsf{CoreXPath}(-)$, using path intersection as an abbreviation:

$$(\mathsf{following}[\mathsf{Image}] \cap (\uparrow^+[\mathsf{Chapter}]/\downarrow^+[\mathsf{Image}])) - (\mathsf{following}[\mathsf{Image}]/\mathsf{following}[\mathsf{Image}])$$

Finally, we give an example of a path expression in $\mathsf{CoreXPath}(*)$. The following path expression, evaluated at the root, retrieves again the first image of each chapter, by repeatedly traveling to the first child and skipping over subtrees that do not contain any image:

$$\downarrow[\mathsf{Chapter}]/\big(\downarrow[\neg\langle\leftarrow\rangle] \cup .[\neg\langle\downarrow^+[\mathsf{Image}]\rangle]/\rightarrow\big)^*[\mathsf{Image}]$$

2.3. STATIC ANALYSIS. We investigate the following problems of static analysis:

—*Path containment*: given two path expressions α, β, is $[\![\alpha]\!]^T_{\mathsf{PExpr}} \subseteq [\![\beta]\!]^T_{\mathsf{PExpr}}$ for all XML trees T?

—*Path satisfiability*: given a path expression α, is there an XML tree T such that $[\![\alpha]\!]^T_{\mathsf{PExpr}} \neq \emptyset$?

—*Node satisfiability*: given a node expression φ, is there an XML tree T such that $[\![\varphi]\!]^T_{\mathsf{NExpr}} \neq \emptyset$?

Each of these can be relativised to documents that satisfy a given schema definition. For example, *path containment w.r.t. EDTDs* means to decide, given two path expressions α, β, and an EDTD D, whether for all XML trees T conforming to D, $[\![\alpha]\!]^T_{\mathsf{PExpr}} \subseteq [\![\beta]\!]^T_{\mathsf{PExpr}}$ holds. Path satisfiability and node satisfiability with respect to EDTDs are defined analogously.

In measuring the complexity of these tasks, we adopt the following definition of *size* for the input: the size of a node expression or path expression is the number of nodes in the syntax tree of that expression, that is, the total number of occurrences of constructors, labels, and atomic path expressions $\rightarrow$, $\leftarrow$, $\uparrow$, $\downarrow$, and ".". The size of an EDTD is the sum of the lengths of the regular expressions assigned to each label, where the length of a regular expression may be measured again by the number of nodes in its syntax tree.

In the following, we establish a tight connection between the three problems that we are studying, and also between the versions with and without EDTD. We start with showing that all three problems (or, to be more accurate, path containment, path *un*satisfiability, and node *un*satisfiability) can be polynomially inter-reduced, both with and without EDTDs. This allows us to state our results only for path containment, but to switch to node satisfiability in proofs when convenient. It is convenient here to consider unsatisfiability rather than satisfiability since the complexity classes involved are not necessarily closed under complementation. We remark that a number of other problems from static analysis, such as *non-empty intersection* considered in Hammerschmidt et al. [2005], are also polynomially inter-reducible with the problems listed above.

PROPOSITION 4. *Let* $L = \mathsf{CoreXPath}_X(Y)$ *for any set of directions* $X \subseteq \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ *and set of operators* $Y \subseteq \{\approx, \cap, -, \mathsf{for}, *\}$. *Then any two of path containment, path unsatisfiability, and node unsatisfiability are polynomially inter-reducible for* L. *Likewise for the EDTD relativised versions of these problems.*

PROOF. We start with the case without EDTDs. Node unsatisfiability polynomially reduces to path unsatisfiability since a node expression φ is satisfiable if, and only if, the path expression $.[\varphi]$ is satisfiable. Path unsatisfiability polynomially reduces to path containment since α is unsatisfiable if, and only if, α is contained in $.[\bot]$. It thus remains to show that path containment can be polynomially reduced to node unsatisfiability.

Let α and β be path expressions, and let Γ be the set of labels occurring in α and β plus one additional label. This choice of Γ is motivated by the fact that (in the absence of EDTDs) whenever α is not contained in β, one can find an XML tree that witnesses the non-containment and in which all nodes have a label from Γ. The reason for this is that if p, p' are any two labels not occurring in α and β, then

occurrences of p in a tree may be freely replaced by p', and vice versa, without affecting the denotation of α or β in the tree (as can be shown by a straightforward formula induction, cf. also Miklau and Suciu [2004]).

Let $\overline{\Gamma} := \Gamma \times \{0, 1\}$, that is, $\overline{\Gamma}$ contains two copies of each label in Γ. Intuitively, a $\overline{\Gamma}$-labeled tree can be seen as a Γ-labeled tree where, in addition, each node is decorated by either 0 or 1. Let $\overline{\alpha}$ be the result of replacing each label p in α with $(p, 0) \vee (p, 1)$, and likewise for $\overline{\beta}$. Moreover, let $\mathbf{1}$ be a shorthand for $\bigvee_{p\in\Gamma}(p, 1)$. Then α is contained in β if, and only if, $\langle\overline{\alpha}[\mathbf{1}]\rangle \wedge \neg\langle\overline{\beta}[\mathbf{1}]\rangle$ is not satisfiable. One direction is easy: if $\langle\overline{\alpha}[\mathbf{1}]\rangle \wedge \neg\langle\overline{\beta}[\mathbf{1}]\rangle$ is satisfiable, then the non-decorated version of any witnessing $\overline{\Gamma}$-labeled XML tree (i.e., the tree obtained by replacing each label of the form (p, i) by p) can be used to show that α is not contained in β. For the other direction, suppose that α is not contained in β. Then there is an XML tree T and a pair of nodes (d, e) belonging to $[\![\alpha]\!]^T_{\mathsf{PExpr}} \setminus [\![\beta]\!]^T_{\mathsf{PExpr}}$. We may assume without loss of generality that all labels in T belong to Γ, that is, the only label that occurs in T but not in α and β is the additional symbol in Γ. Now, let T' be the $\overline{\Gamma}$-tree obtained from T by decorating e with 1 and all other nodes with 0. Then, $d \in [\![\langle\overline{\alpha}[\mathbf{1}]\rangle \wedge \neg\langle\overline{\beta}[\mathbf{1}]\rangle]\!]^{T'}_{\mathsf{NExpr}}$.

This argument can be adapted to the EDTD relativised case: we replace labels occurring in α and β with versions that include an additional decoration from $\{0, 1\}$, and likewise for abstract labels from the EDTD. However, a small technical problem arises: every EDTD dictates a unique label p that occurs at the root of all trees conforming to it, but we would need to replace it with two labels $(p, 0)$ and $(p, 1)$. This issue can be solved by adding a new root to the decorated trees, above the original one, that is simply dropped when translating decorated trees to non-decorated ones. More precisely, given an EDTD $D = (\Delta, P, r, \mu)$, we define $\overline{D} = (\overline{\Delta}, \overline{P}, s, \overline{\mu})$, where

—s is a fresh label,
—$\overline{\Delta} = (\Delta \times \{0, 1\}) \cup \{s\}$,
—$\overline{P}(s) = (r, 0) + (r, 1)$ and for all $p \in \Delta$, $\overline{P}(p, 0) = \overline{P}(p, 1)$ is obtained from $P(p)$ by replacing all atomic subexpressions q with $(q, 0) + (q, 1)$,
—$\overline{\mu}(s) = s$, and for all $p \in \Delta$ and $i \in \{0, 1\}$, $\overline{\mu}(p, i) = (\mu(p), i)$.

Let $\overline{\alpha}$ be the result of replacing each label p in α with $(p, 0) \vee (p, 1)$ and replacing each atomic axis τ with $\tau[\neg s]$. Likewise for $\overline{\beta}$. Then, by similar reasoning as in the case without EDTDs, α is contained in β with respect to the EDTD D if, and only if, $\neg s \wedge \langle\overline{\alpha}[\mathbf{1}]\rangle \wedge \neg\langle\overline{\beta}[\mathbf{1}]\rangle$ is not satisfiable with respect to the EDTD $\overline{D}$. □

The next proposition allows us to transfer lower bounds from the case without EDTDs to the case with EDTDs, and upper bounds in the opposite direction.

PROPOSITION 5. *Let* $L = \mathsf{CoreXPath}_X(Y)$ *for any set of directions* $X \subseteq \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ *and set of operators* $Y \subseteq \{\approx, \cap, -, \mathsf{for}, *\}$. *For each of path containment, path satisfiability, and node satisfiability in* L, *the version without EDTDs polynomially reduces to the EDTD-relativized version.*

PROOF. Due to Proposition 4, it suffices to consider node satisfiability. To reduce satisfiability of φ to node satisfiability with EDTDs, we construct an EDTD that is as nonrestrictive as possible. Let φ be a node expression and let

$\Gamma = \{p_1, \ldots, p_n\}$ be the set of labels occurring in φ plus one additional label. Define $D := (\Delta, P, r, \mu)$, where

—$\Delta = \Gamma \uplus \{s\}$;
—P maps s to the regular expression $p_1 + \cdots + p_n$ and each other label to $(p_1 + \cdots + p_n)^*$;
—μ is the identity function on Δ.

The reason for introducing the new abstract label s is as in the proof of Proposition 4, i.e., we have to deal with EDTDs dictating a fixed label for the root of all trees conforming to it. Let φ' be obtained from φ by replacing every axis $\tau \in \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$ with $\tau[\neg s]$, and likewise for the transitive closures of these axes. We claim that φ is satisfiable if, and only if, $\varphi' \wedge \neg s$ is satisfiable with respect to D. To see this, first assume that φ is satisfiable. Then, there is an XML tree T and a node d in $[\![\varphi]\!]_{\mathsf{NExpr}}$. We may assume that all labels in T belong to Γ. We obtain T' from T by adding a new root, above the original one, labeled with s. Then T' satisfies $\varphi' \wedge \neg s$ and conforms to D. Conversely, let T be a tree that conforms to D and has a node d in $[\![\varphi']\!]_{\mathsf{NExpr}}$. By conformance to D, the root has label s and has a single successor, and the label of each node other than the root node belongs to Γ. We obtain T' from T by cutting off the root. Then, T satisfies φ. □

The following proposition is used for transferring upper bounds from the case without EDTDs to the case with EDTDs, provided the language in question includes all basic axes. The proof is based on an encoding of EDTDs in CoreXPath. It is worth noting that the argument does not require the presence of the transitive closure operator in the language.

PROPOSITION 6. *Let* $L = \mathsf{CoreXPath}(X)$ *for any* $X \subseteq \{\approx, \cap, -, \mathsf{for}, *\}$. *For each of* path containment, path satisfiability, *and* node satisfiability *in* L, *the EDTD-relativised version polynomially reduces to the version without EDTDs.*

PROOF. Due to Proposition 4, it suffices to consider node satisfiability. Thus, let φ be a node expression and $D = (\Delta, P, r, \mu)$ an EDTD. By standard techniques, we may construct in polynomial time an equivalent nondeterministic finite state automaton (NFA) A_t for each regular expression $P(t)$, $t \in \Delta$. We may assume without loss of generality that the sets of states Q_{A_t} of these automata are disjoint.

We now introduce the notion of a *witness tree*, whose purpose is to witness that a certain XML tree conforms to D. The node labels in a witness tree are from $\Gamma := \Delta \times \bigcup_{t \in \Delta} Q_{A_t}$. Each witness tree $T = (N, R_\downarrow, R_\rightarrow, L)$ has to satisfy the following conditions, where we use $L^1(n)$ to denote the first component of $L(n)$ and $L^2(n)$ for the second component, for all $n \in N$:

(1) if $n \in N$ is the root of T, then $L^1(n) = r$;
(2) for each node n with (ordered) children $n_1, \ldots, n_k$, there is a final state q of the automaton $A_{L^1(n)}$ such that the sequence of states $L^2(n_1), \ldots, L^2(n_k), q$ is an accepting run of $A_{L^1(n)}$ on $L^1(n_1), \ldots, L^1(n_k)$;
(3) for each leaf n, the automaton $A_{L^1(n)}$ accepts the empty word.

Note that $L^2(n)$ may be arbitrary if n is the root node.

The existence of a witness tree $T = (N, R_\downarrow, R_\rightarrow, L)$ shows that the corresponding XML tree $T' = (N, R_\downarrow, R_\rightarrow, L')$, with $L'(n) = \mu(L^1(n))$ for all $n \in N$, conforms

to D. Conversely, every XML tree T' that conforms to D gives rise to an isomorphic witness tree whose node labels can be constructed from the function L' from Definition 2 and suitable accepting runs of the involved NFAs.

We construct a node expression ψ, using labels from Γ, such that ψ is true at the root of an XML tree T if, and only if, T is a witness tree. More precisely, ψ is a conjunction that has five conjuncts: one for condition (1) of witness trees, one for condition (3), and three for condition (2) stating that every run starts in an initial state, ends in a final state, and respects the transition relation. For example, to capture that the run respects the transition relation, we have to express the following: whenever a node is labeled (p, q) and has a parent labeled (p', q') and next sibling labeled (p'', q''), the transition relation $\delta_{A_{p'}}$ of the automaton $A_{p'}$ contains the triple (q, p, q''), that is,

$$\bigwedge_{\substack{(p,q),(p',q'),(p'',q'')\in\Gamma \\ \text{with } (q,p,q'')\notin\delta_{A_{p'}}}} \neg\langle\downarrow^*[(p', q')]/\downarrow[(p, q)]/\rightarrow[(p'', q'')]\rangle .$$

Since we work with NFAs instead of with regular expressions, full transitive closure is not needed in ψ; indeed, ψ is a CoreXPath node expression.

We are interested in the existence of a witness tree whose corresponding XML tree satisfies the input expression φ. Thus, we take the conjunction of ψ with a formula expressing that φ holds somewhere in the corresponding XML tree. More precisely, let φ' be obtained from φ by replacing each label p by the disjunction of all labels in $(p', q) \in \Gamma$ such that $\mu(p') = p$. Then we have that φ is satisfiable with restect to D if, and only if, $\psi \wedge \neg\langle\uparrow\rangle \wedge \langle\downarrow^*[\varphi']\rangle$ is satisfiable. □

Thus, all upper bounds in Table I except the one for $\mathsf{CoreXPath}_{\downarrow}(\cap)$ carry over from the unrestricted case to the EDTD-relativised case. The $\mathsf{CoreXPath}_{\downarrow}(\cap)$ case is not captured since, in Proposition 6, we assume the presence of all axes. For this reason, the EXPSPACE-upper bound for $\mathsf{CoreXPath}_{\downarrow}(\cap)$ will be shown directly with EDTDs.

3. CoreXPath($*, \approx$) *Is in ExpTime*

We prove that path containment for CoreXPath($*, \approx$) is in EXPTIME using two-way alternating tree automata.

3.1. PREPARATION.

To simplify the proof, we work with a different, but equally expressive version of CoreXPath($*, \approx$). This version differs in four aspects from the original one:

(1) We replace path equalities with node expressions of the form $\mathsf{loop}(\alpha)$, which test whether a node is reachable from itself along α. Note that $\mathsf{loop}(\alpha)$ can be expressed as "$\alpha \approx .$". Conversely, $\alpha \approx \beta$ can be written as $\mathsf{loop}(\alpha/\beta^{\smile})$, where $\beta^{\smile}$ is the converse of β defined inductively as follows:

$$\begin{array}{lll} \uparrow^{\smile} & = & \downarrow \\ \downarrow^{\smile} & = & \uparrow \\ \leftarrow^{\smile} & = & \rightarrow \\ \rightarrow^{\smile} & = & \leftarrow \\ .^{\smile} & = & . \end{array} \qquad \begin{array}{lll} (\alpha/\beta)^{\smile} & = & (\beta^{\smile}/\alpha^{\smile}) \\ (\alpha \cup \beta)^{\smile} & = & \alpha^{\smile} \cup \beta^{\smile} \\ (\alpha[\varphi])^{\smile} & = & .[\varphi]/(\alpha^{\smile}) \\ (\alpha^*)^{\smile} & = & (\alpha^{\smile})^* \end{array}$$

(2) We drop node expressions $\langle\alpha\rangle$. This can be done without loss of generality since $\langle\alpha\rangle$ can be expressed as $\mathsf{loop}(\alpha/\uparrow^*/\downarrow^*)$.

(3) We replace the vertical axes $\downarrow$ and $\uparrow$ with the *first-child* axis $\downarrow_1$ and its converse $\uparrow_1$. This can be done without loss of generality since $\downarrow$ can be expressed as $\downarrow_1/\rightarrow^*$ and, conversely, $\downarrow_1$ can be written as $\downarrow[\neg\langle\leftarrow\rangle]$. Likewise for $\uparrow$ and $\uparrow_1$.

(4) We replace path expressions with NFAs (non-deterministic finite word automata) whose alphabet is comprised of the *basic axes* $\downarrow_1, \uparrow_1, \leftarrow, \rightarrow$ and filter expressions of the form $.[\varphi]$. This is justified by the observation that path expressions of $\mathsf{CoreXPath}(*, \approx)$ are regular expressions over this alphabet if we replace subexpressions of the form $\alpha[\varphi]$ with $\alpha/.[\varphi]$. We call such NFAs *path automata*.

Formally, the resulting version of CoreXPath, which we call $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$, is defined as follows.

Definition 7 ($\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$). The node expressions and path automata of $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ are defined by simultaneous induction:

—Node expressions are built according to the following syntax rule, where p ranges over labels from Σ, π over path automata, and φ, ψ over node expressions: $\varphi ::= p \mid \mathsf{loop}(\pi) \mid \top \mid \neg\varphi \mid \varphi \wedge \psi$

—A *path automaton* is a tuple $\pi = (Q, \Delta, q_I, q_F)$, where Q is a finite set of states, q_I and q_F are the initial and final state, and Δ is a finite subset of

$$Q \times \big(\{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow\} \cup \{.[\psi] \mid \psi \text{ is a node expression}\}\big) \times Q,$$

The semantics is as before (cf. Table II), with

$$[\![\mathsf{loop}(\alpha)]\!]^T_{\mathsf{NExpr}} = \big\{n \mid (n, n) \in [\![\alpha]\!]^T_{\mathsf{PExpr}}\big\}$$

and, for any path automaton $\pi = (Q, \Delta, q_I, q_F)$, $[\![\pi]\!]^T_{\mathsf{PExpr}}$ is the relation containing all pairs (n, m) for which there is a sequence $n = n_0, \ldots, n_k = m$ of nodes and a word $\alpha_0 \cdots \alpha_k$ in the language recognized by π such that for all $i < m$:

—either $\alpha_i = .[\varphi]$, $n_i = n_{i+1}$ and $n_i \in [\![\varphi]\!]^T_{\mathsf{NExpr}}$
—or $\alpha_i \in \{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow\}$ and $(n_i, n_{i+1}) \in R_{\alpha_i}$

(where $R_{\downarrow_1}$ and $R_{\uparrow_1}$ are defined in the obvious way). The sequence $n_0, \ldots, n_m$ is called a π*-trace* from n to m.

Using the equivalences stated in (1) to (4) above and the usual translation from regular expressions to NFAs, every $\mathsf{CoreXPath}(*, \approx)$ path expression can be translated in linear time to an equivalent $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ path expression. Note that since "skip" transitions can be defined as $.[\top]$, it is enough to have only a single final state in NFAs.

The *size* of a $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ node expression φ and path automaton π is denoted by $|\varphi|$ and $|\pi|$, and defined inductively: $|p| = |\top| = 1$, $|\neg\varphi| = |\varphi| + 1$, $|\varphi \wedge \psi| = |\varphi| + |\psi| + 1$, $|\mathsf{loop}(\pi)| = |\pi| + 1$, and $|\pi| = |Q| + \sum_{(q,.[\varphi],q') \in \Delta} |\varphi|$ if $\pi = (Q, \Delta, q_I, q_F)$.

For a path automaton $\pi = (Q, \Delta, q_I, q_F)$ and states $q, q' \in Q$, we use $\pi_{q,q'}$ as shorthand for (Q, Δ, q, q').

3.2. ALTERNATING TREE AUTOMATA. For our decision procedure, we use a slightly non-standard version of two-way alternating parity tree automata (2ATAs), which work on XML trees and can move along basic axes. In contrast, standard 2ATAs work on infinite ranked trees and cannot move to siblings. Our 2ATAs traverse a tree using the BASIC-STEPS $=\{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow, \varepsilon\}$, where the first four steps correspond to the basic axes of $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ and ε means staying at the current node. For each node n of an XML tree, POSS-STEPS(n) denotes the set of basic steps that can be performed from n (which always includes ε). For each $a \in$ POSS-STEPS(n), we will denote by $n \cdot a$ the node reached from n by performing the basic step a. In the following definition, $\wp()$ will denote powerset, and $\mathcal{B}^+(X)$ will denote the set of all positive Boolean formulas over variables from X, including true and false.

Definition 8 (*2ATA*). A *two-way alternating tree automaton (2ATA)* is a tuple $A = (Q, \delta, q_0, Acc)$, where

—Q is a finite set of states

—$\delta : (Q \times \Sigma \times \wp(\text{BASIC-STEPS})) \rightarrow \mathcal{B}^+(\text{BASIC-STEPS} \times Q)$ is the transition function. We require that all basic steps occurring in $\delta(q, \sigma, S)$ belong to S.

—q_0 is the initial state

—$Acc : Q \rightarrow \mathbb{N}$ specifies a parity acceptance condition.

Intuitively, if $\delta(q, \sigma, S) = \psi$, this means that when the automaton is in state q, reads σ, and the current node allows exactly the basic steps in S, then the transition is as described by ψ, in the usual sense of alternating automata, see, for example, Vardi [1998]. Note that the alphabet Σ of 2ATAs is the same as the alphabet underlying XML trees. Formally, the semantics is given in terms of runs.

Definition 9 (*Run*). A *run* of a 2ATA $A = (Q, \delta, q_0, Acc)$ on an XML tree $T = (N, R_\downarrow, R_\rightarrow, L)$ is an (unordered and not necessarily finite) tree $r = (V, E, \ell)$, where ℓ is a function that labels nodes with pairs $(n, q) \in N \times Q$, and such that the following conditions hold:

—Initial state: The root of r is labeled by (n, q_0), where n is the root of T.

—Transition function: If $\ell(x) = (n, q)$, and $\delta(q, L(n), \text{POSS-STEPS}(n)) = \theta$, then there is a set $S \subseteq (\text{POSS-STEPS}(n) \times Q)$ that satisfies θ and such that for each $(a, q) \in S$, x has a child y with $\ell(y) = (n \cdot a, q)$.

The run r is *accepting* if for each infinite path in r, the lowest number assigned by Acc to a state occurring infinitely often on that path is even. The 2ATA A *accepts* T if there is an accepting run of A on T. $\mathcal{L}(A)$ is the set of XML trees accepted by A.

It is shown in Vardi [1998] that the emptiness problem for two-way alternating automata on infinite binary trees is decidable in EXPTIME. This result easily carries over to our version of 2ATAs.

THEOREM 10. *Given a 2ATA A, it is decidable in* EXPTIME *whether $\mathcal{L}(A)$ is empty.*

PROOF SKETCH. We sketch how to polynomially reduce emptiness of 2ATAs (in our sense) to emptiness of standard 2ATAs on infinite binary trees. By an infinite binary tree, we mean an infinite tree where every node has precisely two children

(the first child and the second child). In particular, these trees have no leafs. The definition of a two-way alternating automaton on infinite binary trees differs from one for XML trees in that the transition function does not need to take into account the set of possible steps from the current node. Thus, the domain of the transition function of a 2ATA on infinite binary trees is of the form $Q \times \Sigma$ rather than $Q \times \Sigma \times \wp(\text{BASIC-STEPS})$. In our reduction, we bridge these differences by making the set of possible steps part of the label of the node, and by adding additional nodes to the tree in order to turn it into an infinite tree.

More precisely, for an XML tree $T = (N, R_\downarrow, R_\rightarrow, L)$, the *deco version* T^d of T is the infinite binary tree defined as follows. First, we see T as a finite binary tree, by viewing $R_{\downarrow_1}$ and $R_\rightarrow$ as a *first child* and *second child* relation. We then extend this finite binary tree to an infinite binary tree by adding new nodes, so that every node has two children in the new tree. Finally, we change the labeling function, so that it assigns to each node $n \in N$ the pair $(L(n), \text{POSS-STEPS}(n))$ and to each new node a new node label $\perp$. The tree $T^d = (N', R_1, R_2, L^d)$ thus obtained is an infinite binary tree over the alphabet $\Sigma^d := (\Sigma \times 2^{\text{BASIC-STEPS}}) \cup \{\perp\}$.

The same representation can be used on the side of the automaton: given a 2ATA A on XML trees, we can convert A into a 2ATA A_1 on infinite binary trees, such that A_1 accepts exactly the deco versions of trees accepted by A. Hence, testing emptiness of A reduces to testing emptiness of A_1. Roughly, the automaton A_1 is constructed by taking a copy of A where each transition $\delta(q, \sigma, S) = \psi$ is replaced by a transition $\delta(q, (\sigma, S)) = \psi$, and then intersecting the automaton with automata that check that (i) the sets of possible steps encoded by the labels of the nodes of the deco tree are consistent, (ii) whenever a node is labeled with $\perp$, then so are all its descendants, and (iii) on every path, the $\perp$ symbol eventually occurs. Note that intersection is trivial for alternating automata: to intersect 2ATAs A_1, A_2 with initial states q_0 and q_0', respectively, it suffices to take the disjoint union of A_1 and A_2, and to add a new initial state such that for any symbol and set of possible steps, the transition formula is $(\varepsilon, q_0) \wedge (\varepsilon, q_0')$. □

3.3. THE REDUCTION. Our aim is to translate each $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ node expression φ into a 2ATA A_φ such that A_φ accepts precisely the XML trees that satisfy φ at some node. To prepare for this translation, we give an inductive characterization of the node sets defined by expressions of the form $\mathsf{loop}(\alpha)$.

LEMMA 11. *Let $\pi = (Q, \Delta, q_I, q_F)$ be a path automaton, and $T = (N, R_\downarrow, R_\rightarrow, L)$ an XML tree. Define* $\text{LOOPS}_\pi := \bigcup_{s \geq 0} \text{LOOPS}_\pi^{(s)}$ *where*

$$\begin{aligned}\text{LOOPS}_\pi^{(0)} = {} & \{(n, q, q) \mid n \in N, q \in Q\} \cup \\ & \{(n, q_i, q_j) \mid (q_i, .[\varphi], q_j) \in \Delta \text{ and } n \in [\![\varphi]\!]^T_{\mathsf{NExpr}}\}.\end{aligned}$$

and, for all $s > 0$, $\text{LOOPS}_\pi^{(s)}$ *is* $\text{LOOPS}_\pi^{(s-1)}$ *extended with*

(1) (n, q_i, q_ℓ) *if* $nR_\tau m$ *with* $\tau \in \{\downarrow_1, \uparrow_1, \leftarrow, \rightarrow\}$, $(m, q_j, q_k) \in \text{LOOPS}_\pi^{(s-1)}$, *and* $(q_i, \tau, q_j), (q_k, \tau^\smile, q_\ell), \in \Delta$;

(2) (n, q_i, q_k) *if* $(n, q_i, q_j) \in \text{LOOPS}_\pi^{(s-1)}$ *and* $(n, q_j, q_k) \in \text{LOOPS}_\pi^{(s-1)}$.

Then $(n, q, q') \in \text{LOOPS}_\pi$ *if, and only if,* $n \in [\![\mathsf{loop}(\pi_{q,q'})]\!]^T_{\mathsf{NExpr}}$.

TABLE III. TRANSITION FUNCTION OF THE 2ATA A_φ

$$
\begin{array}{ll}
\delta(\ell, q_p, \text{POSS-STEPS}) & = \top \text{ if } \ell = p, \bot \text{ otherwise} \\
\delta(\ell, q_{\neg p}, \text{POSS-STEPS}) & = \bot \text{ if } \ell = p, \top \text{ otherwise} \\
\delta(\ell, q_{\psi\wedge\chi}, \text{POSS-STEPS}) & = (\varepsilon, q_\psi) \wedge (\varepsilon, q_\chi) \\
\delta(\ell, q_{\neg(\psi\wedge\chi)}, \text{POSS-STEPS}) & = (\varepsilon, q_{\neg\psi}) \vee (\varepsilon, q_{\neg\chi}) \\
\delta(\ell, q_{\mathsf{loop}(\pi_{q_i,q_j})}, \text{POSS-STEPS}) & = \top \text{ if } q_i = q_j, \text{ otherwise} \\
 & \quad \bigvee_{(q_i,.[\chi],q_j)\in\Delta} (\varepsilon, q_\chi) \\
 & \quad \vee \bigvee_{(q_i,\tau,q_k),(q_\ell,\tau^\smile,q_j)\in\Delta,\tau\in\text{POSS-STEPS}} (\tau, q_{\mathsf{loop}(\pi_{q_k,q_\ell})}) \\
 & \quad \vee \bigvee_{q_k\in Q} ((\varepsilon, q_{\mathsf{loop}(\pi_{q_i,q_k})}) \wedge (\varepsilon, q_{\mathsf{loop}(\pi_{q_k,q_j})})) \\
\delta(\ell, q_{\neg\mathsf{loop}(\pi_{q_i,q_j})}, \text{POSS-STEPS}) & = \bot \text{ if } q_i = q_j, \text{ otherwise} \\
 & \quad \bigwedge_{(q_i,.[\chi],q_j)\in\Delta} (\varepsilon, q_{\neg\chi}) \\
 & \quad \wedge \bigwedge_{(q_i,\tau,q_k),(q_\ell,\tau^\smile,q_j)\in\Delta,\tau\in\text{POSS-STEPS}} (\tau, q_{\neg\mathsf{loop}(\pi_{q_k,q_\ell})}) \\
 & \quad \wedge \bigwedge_{q_k\in Q} ((\varepsilon, q_{\neg\mathsf{loop}(\pi_{q_i,q_k})}) \vee (\varepsilon, q_{\neg\mathsf{loop}(\pi_{q_k,q_j})}))
\end{array}
$$

PROOF. The "only if" direction is the easy direction: it can be shown by induction on s that $(n, q, q') \in \text{LOOPS}_\pi^{(s)}$ implies $n \in [\![\mathsf{loop}(\pi_{q,q'})]\!]^T_{\mathsf{NExpr}}$, using the semantics.

For the "if" direction, let $n \in [\![\mathsf{loop}(\pi_{q,q'})]\!]^T_{\mathsf{NExpr}}$. Then there is a sequence $t = n_0, \ldots, n_k \in N$ with $n = n_0 = n_k$ and a word $w = \alpha_0 \cdots \alpha_{k-1}$ in the language recognized by π such that for all $i < k$,

—either $\alpha_i = .[\varphi]$, $n_i = n_{i+1}$, and $n_i \in [\![\varphi]\!]^T_{\mathsf{NExpr}}$
—or $\alpha_i \in \{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow\}$ and $(n_i, n_{i+1}) \in R_{\alpha_i}$.

We prove by induction on k that the existence of such t and w implies that $(n, q, q') \in \text{LOOPS}_\pi$. The case $k = 0$ is trivial by definition of $\text{LOOPS}_\pi^{(0)}$. If $k = 1$, then $\alpha_0 = .[\varphi]$ for some node expression φ and again we get $(n, q, q') \in \text{LOOPS}_\pi^{(0)}$. Now let $k \geq 2$. First assume that there is an $i \in \{1, \ldots, k-1\}$ such that $n_i = n$. Then we split t into $n_0, \ldots, n_i$ and $n_i, \ldots, n_k$ and likewise for w, use the induction hypothesis and (2). Now assume that there is no such i. Since we use the functional relations $\downarrow_1, \uparrow_1, \rightarrow, \leftarrow$ as our basic axes, we then have $\alpha_0 = \alpha_{k-1}^\smile \in \{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow\}$ and $n_1 = n_{k-1}$. It remains to apply the induction hypothesis and (1). □

We are now ready to convert a $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ node expression φ into a 2ATA A_φ. Let $\varphi' = \mathsf{loop}(\downarrow^*[\varphi]/\uparrow^*)$, that is, φ is satisfiable if, and only if, φ' is satisfiable at the root of an XML tree. Let $\mathsf{cl}(\varphi')$ be the smallest set of node expressions containing φ' and such that

—$\mathsf{cl}(\varphi')$ is closed under taking subexpressions;
—$\mathsf{cl}(\varphi')$ is closed under single negations (i.e., whenever $\psi \in \mathsf{cl}(\varphi')$ and ψ is not already of the form $\neg\psi'$, then $\neg\psi \in \mathsf{cl}(\varphi')$); and
—for all $\mathsf{loop}(\pi) \in \mathsf{cl}(\varphi')$ and q_k, q_ℓ states of π, $\mathsf{loop}(\pi_{q_k,q_\ell})$ also belongs to $\mathsf{cl}(\varphi')$.

The cardinality of $\mathsf{cl}(\varphi')$ and the size of its elements (as defined in Section 3.1) is bounded polynomially in the size of φ. We now define $A_\varphi := (Q, \delta, q_0, Acc)$, where $Q = \{q_\psi \mid \psi \in \mathsf{cl}(\varphi')\}$, δ is defined as in Table III, $q_0 = q_{\varphi'}$, Acc assigns 1 to all states of the form $q_{\mathsf{loop}(\pi_{q_i,q_j})}$, and 2 to all others. In Table III, POSS-STEPS

ranges over all subsets of BASIC-STEPS, Δ is the transition relation of π_{q_i,q_j}, and the case $q_{\mathsf{loop}}(\pi)$ is identified with $q_{\mathsf{loop}}(\pi_{q_I,q_F})$, with q_I and q_F the initial and final state of π. The chosen parity condition ensures that no state of the form $q_{\mathsf{loop}(\pi_{q_i,q_j})}$ may occur forever from some point onwards on a path in the run. This is necessary to guarantee that the looping path automaton π will not forever delay its return to the node where the loop originated.

LEMMA 12. *For all XML trees T, $T \in \mathcal{L}(A_\varphi)$ if, and only if, $[\![\varphi]\!]^T_{\mathsf{NExpr}} \neq \emptyset$.*

PROOF. It suffices to show that $T = (N, R_\downarrow, R_\rightarrow, L) \in \mathcal{L}(A_\varphi)$ if, and only if, T satisfies φ' at the root. For the "if" direction, we inductively construct a run $r = (V, E, \ell)$ of A_φ on T such that for all $x \in V$, $\ell(x) = (n, q_\psi)$ implies $n \in [\![\psi]\!]^T_{\mathsf{NExpr}}$. We start with r consisting of a single root node labeled $(n, q_{\varphi'})$, where n is the root of T. For the induction step, assume that $x \in V$ with $\ell(x) = (n, q_\psi)$ and $\delta(L(n), q_\psi, \text{POSS-STEPS}(n)) \notin \{\top\}$. If $\psi = \vartheta \wedge \chi$, then we add two successors y and y' of x and set $\ell(y) = (n, q_\vartheta)$ and $\ell(y') = (n, q_\chi)$. The case $\psi = \neg(\vartheta \wedge \chi)$ reflects the definition of δ in an analogous way. Now let $\psi = \mathsf{loop}(\pi_{q_i,q_j})$, P be the states of π, and Δ the transition relation. By definition of δ and since $\delta(L(n), q_\psi, \text{POSS-STEPS}(n)) \neq \top$, we have $q_i \neq q_j$. Since $n \in [\![\psi]\!]^T_{\mathsf{NExpr}}$ and by Lemma 11, we have that $(n, q_i, q_j) \in \text{LOOPS}^{(s)}_\pi$ for some $s \geq 0$, and one of the following three cases applies:

(1) $(q_i, .[\chi], q_j) \in \Delta$ and $n \in [\![\chi]\!]^T_{\mathsf{NExpr}}$. Then, we add a new successor y of x with $\ell(y) = (n, q_\chi)$.
(2) There is $m \in N, q_k, q_\ell \in P$, and $\tau \in \{\downarrow_1, \uparrow_1, \leftarrow, \rightarrow\}$ with $nR_\tau m$, $(m, q_k, q_\ell) \in \text{LOOPS}^{(s-1)}_\pi$, and $(q_i, \tau, q_k), (q_\ell, \tau^\smile, q_j) \in \Delta$. Then we add a successor y of x with $\ell(y) = (m, q_{\mathsf{loop}(\pi_{q_k,q_\ell})})$.
(3) There is a $q_k \in P$ with $(n, q_i, q_k), (n, q_k, q_j) \in \text{LOOPS}^{(s-1)}_\pi$. Then we add two successors y and y' of x with $\ell(y) = (n, q_{\mathsf{loop}(\pi_{q_i,q_k})})$ and $\ell(y') = (n, q_{\mathsf{loop}(\pi_{q_k,q_j})})$.

The case $\psi = \neg\mathsf{loop}(\pi_{q_i,q_j})$ is treated dually, c.f. the definition of δ. It can be verified that the constructed run is an accepting run of A_φ on T.

For the "only if" direction, suppose there is an accepting run r of A_φ on T. It suffices to prove that if (n, q_ψ) occurs as a label in r, then $n \in [\![\psi]\!]^T_{\mathsf{NExpr}}$. The proof is by induction on ψ with restect to the smallest transitive relation "$\prec$" on $\mathsf{cl}(\varphi')$ that satisfies the following conditions for all $\vartheta \in \mathsf{cl}(\varphi')$:

—$\chi \prec \vartheta$ for all strict subexpressions χ of ϑ;
—$\neg\chi \prec \vartheta$ for all strict subexpressions χ of ρ if $\vartheta = \neg\rho$.

We only do the interesting two cases explicitly, starting with $\psi = \mathsf{loop}(\pi_{q_i,q_j})$. We assign to each node x in r a rank $\delta(x)$ according to the number of consecutive occurrences of labels of the form $(m, q_{\mathsf{loop}(\pi_{q,q'})})$ from x onwards. More precisely, $\delta(x)$ is the length of the largest sequence of nodes $(x_1, \ldots, x_k)$ each having a label of the form $(m, q_{\mathsf{loop}(\pi_{q,q'})})$ such that $x_1 = x$ and $x_i R_\downarrow x_{i+1}$ for all $i < k$ (note that $\delta(x)$ can be 0). Due to the acceptance condition, $\delta(x)$ is always finite. We prove by induction on $\delta(x)$ that if x is labeled $(n, q_{\mathsf{loop}(\pi_{q_i,q_j})})$, then $n \in [\![\mathsf{loop}(\pi_{q_i,q_j})]\!]^T_{\mathsf{NExpr}}$. If $\delta(x) = 1$, then the definition of δ yields that $q = q'$ or there is a $(q_i, .[\chi], q_j) \in \Delta$ and a successor y of x with label (n, q_χ). In the former case, we are done. In

the latter, we apply the (outer) induction hypothesis and the semantics. Now let $\delta(x) > 1$. By definition of δ, one of the following two cases applies:

(a) there are $(q_i, \tau, q_k), (q_\ell, \tau^\smile, q_j) \in \Delta$, $\tau \in$ POSS-STEPS, and a successor y of x with label $(m, q_{\mathsf{loop}(\pi_{q_k,q_\ell})})$ and $nR_\tau m$;

(b) there are $q_k \in Q$ and successors y and y' of x with labels $(n, q_{\mathsf{loop}(\pi_{q_i,q_k})})$ and $(n, q_{\mathsf{loop}(\pi_{q_k,q_j})})$.

In both cases, it suffices to apply the (inner) induction hypothesis and the semantics.

Now for the case $\psi = \neg\mathsf{loop}(\pi_{q_i,q_j})$. Assume towards a contradiction that there is a node x in r with label $(n, q_{\neg\mathsf{loop}(\pi_{q_i,q_j})})$ and $n \in [\![\mathsf{loop}(\pi_{q_i,q_j})]\!]^T_{\mathsf{NExpr}}$. By Lemma 11, $n \in \text{LOOPS}^{(s)}_\pi$ for some $s \geq 0$. Choose an x with the above properties such that s is minimal. If $s = 0$, we have $q_i = q_j$ or there is a $(q_i, .[\chi], q_j) \in \Delta$ with $n \in [\![\chi]\!]^T_{\mathsf{NExpr}}$. In the former case, we have $\delta(L(n), q_{\neg\mathsf{loop}(\pi_{q_i,q_j})}) = \bot$, in contradiction to the existence of r. For the latter case, we note that, by definition of δ, x has a successor y with label $(n, q_{\neg\chi})$. Thus the (outer) induction hypothesis yields $n \in [\![\neg\chi]\!]^T_{\mathsf{NExpr}}$, which is a contradiction. Now let $s > 0$. Then one of the following two cases applies:

(a) there are $m \in N$, $\tau \in \{\downarrow_1, \uparrow_1, \leftarrow, \rightarrow\}$, and $q_k, q_\ell \in Q$ with $nR_\tau m$, $(m, q_k, q_\ell) \in \text{LOOPS}^{(s-1)}_\pi$, and $(q_i, \tau, q_k), (q_\ell, \tau^\smile, q_j), \in \Delta$;

(b) there is $q_k \in Q$ with $(n, q_i, q_k) \in \text{LOOPS}^{(s-1)}_\pi$ and $(n, q_k, q_j) \in \text{LOOPS}^{(s-1)}_\pi$.

We only consider case (a), as case (b) is similar. By definition of δ, there is a successor y of x in r with label $(m, q_{\neg\mathsf{loop}(\pi_{q_k,q_\ell})})$. By minimality of i, $m \in [\![\neg\mathsf{loop}(\pi_{q_k,q_\ell})]\!]^T_{\mathsf{NExpr}}$. By Lemma 11, this is a contradiction to $(m, q_k, q_\ell) \in \text{LOOPS}^{(s-1)}_\pi$. □

By Theorem 10 and Lemma 12 and since all components of $\mathcal{A}_\varphi$ are of size polynomial in $|\varphi|$, we obtain the following result.

THEOREM 13. *Satisfiability of* $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ *node expressions is in* EXPTIME.

By Proposition 4, together with the facts that (i) there is a linear translation from $\mathsf{CoreXPath}(*, \approx)$ to $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ and (ii) path containment for CoreXPath [Benedikt et al. 2008] is EXPTIME-hard, we thus obtain

COROLLARY 14. *Path containment for* $\mathsf{CoreXPath}(*, \approx)$ *is* EXPTIME*-complete.*

By Proposition 6, the same holds for path containment relative to an EDTD.

4. $\mathsf{CoreXPath}(*, \cap)$ *is in 2-ExpTime*

We show that path containment for $\mathsf{CoreXPath}(*, \cap)$ is decidable in 2-EXPTIME using a satisfiability-preserving and exponential translation to $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$. If the nesting depth of path intersection is bounded by a constant, the translation becomes

polynomial. It follows that, in this case, path containment for CoreXPath($*, \cap$) is only EXPTIME-complete.

The translation consists of two steps. First, we give an exponential and equivalence-preserving translation of CoreXPath($*, \cap$) expressions into CoreXPath$_{\mathsf{NFA}}$($*$, loop) extended with a let construct, which allows to introduce abbreviations for node expressions. Second, we show that the let construct can be eliminated in polynomial time while preserving (un)satisfiability of expressions. In Section 8, we (re)use the first step to derive an upper bound on the relative succinctness of CoreXPath($*, \cap$) compared to CoreXPath($*, \approx$).

4.1. ADDING A let CONSTRUCT TO CoreXPath$_{\mathsf{NFA}}$($*$, loop). We use CoreXPath$_{\mathsf{NFA}}$ ($*$, loop, let) to denote the extension of CoreXPath$_{\mathsf{NFA}}$($*$, loop) with a let construct. In its simplest form, this construct allows to build node expressions of the form "let $p := \varphi$ in ψ," with $p \in \Sigma$ a label. This let expression introduces p as an abbreviation for φ in ψ, that is, the expression is equivalent to ψ with the label p uniformly replaced with φ. Observe that labels p, p' that are bound by let behave slightly different than usual: depending on the formulas that p and p' abbreviate, $p \wedge p'$ is not necessarily unsatisfiable if $p \neq p'$. The let construct clearly does not add any expressive power, but it makes expressions exponentially more succinct.

In CoreXPath$_{\mathsf{NFA}}$($*$, loop, let), we admit a generalized version of let that admits sequences of bindings. An *environment* is a sequence $\rho = (p_1, \varphi_1), \ldots, (p_n, \varphi_n)$, where the first component of each pair is an atomic label, and the second is a node expression. We use "let ρ in ψ" as shorthand for "let $p_1 := \varphi_1$ in let $p_2 := \varphi_2 \ldots$ in ψ".

Path automata of CoreXPath$_{\mathsf{NFA}}$($*$, loop, let) are defined as for CoreXPath$_{\mathsf{NFA}}$($*$, loop) except that, of course, they may contain tests on CoreXPath$_{\mathsf{NFA}}$($*$, loop, let) node expressions. An *extended path automaton* ("*EPA*") is a pair (π, ρ), with π a CoreXPath$_{\mathsf{NFA}}$($*$, loop, let) path automaton and ρ an environment. It is simply a succinct representation for π^ρ, the path automaton obtained from π by replacing all labels bound in ρ by their definition (in the appropriate order).

The *size* of node expressions and (extended) path automata is defined by simultaneous recursion, as in Section 3.1, where $|\text{let } p := \varphi \text{ in } \psi| = |\varphi| + |\psi| + 1$. The size of an environment $\rho = ((p_1, \varphi_1), \ldots, (p_n, \varphi_n))$ is $\Sigma_{i \leq n}(|\varphi_i| + 1)$ and $|\text{let } \rho \text{ in } \psi| = |\rho| + |\psi|$. The size of an EPA (π, ρ) is simply $|\pi| + |\rho|$.

4.2. TRANSLATION FROM CoreXPath($*, \cap$) TO CoreXPath$_{\mathsf{NFA}}$($*$, loop, let). The following lemma provides the basis of the translation from CoreXPath($*, \cap$) to CoreXPath$_{\mathsf{NFA}}$($*$, loop, let). In the following, we use $|\pi|_S$ to denote the number of states of the (extended) path automaton π.

LEMMA 15. *For all* CoreXPath$_{\mathsf{NFA}}$($*$, loop, let) *EPAs* (π_1, ρ_1) *and* (π_2, ρ_2), *there is a* CoreXPath$_{\mathsf{NFA}}$($*$, loop, let) *EPA* $(\pi^\cap, \rho^\cap)$ *that is equivalent to* ${\pi_1}^{\rho_1} \cap {\pi_2}^{\rho_2}$ *and*

(1) $|\pi^\cap|_S = |\pi_1|_S \cdot |\pi_2|_S$,
(2) $|\pi^\cap| = |\pi_1|_S^2 \cdot |\pi_2|_S + |\pi_1|_S \cdot |\pi_2|_S^2 + |\pi_1|_S \cdot |\pi_2|_S$,
(3) $|\rho^\cap| \leq |\rho_1| + |\rho_2| + |\pi_1|_S^2 \cdot (|\pi_1| + 2) + |\pi_2|_S^2 \cdot (|\pi_2| + 2)$.

PROOF. Let $\pi_1 = (Q, \Delta, q_I, q_F)$ and $\pi_2 = (Q', \Delta', q_I', q_F')$, and assume without loss of generality that the sets of atomic labels bound by ρ_1 and ρ_2 are disjoint. The idea for defining $\pi^\cap$ is as follows. Suppose that m and n are nodes in an XML

tree T such that $(m, n) \in [\![\pi]\!]^T_{\mathsf{PExpr}} \cap [\![\pi']\!]^T_{\mathsf{PExpr}}$. This is witnessed by a π-trace t from m to n and a π'-trace t' from m to n. We want the automaton $\pi^\cap$ to check the existence of both traces, but face the problem that t and t' may follow different routes through T. Clearly, there is a unique cycle-free path p from m to n in T and both t and t' must travel along p, but may additionally make loops that return to the same point on p where they diverted. Thus, $\pi^\cap$ can verify the existence of the two traces by travelling along p and using loop to cut short all loops of t and t' that diverge from p.

To implement this idea, we define $\pi^\cap$ as $(Q \times Q', \Delta^\cap, \langle q_0, q'_0\rangle, \langle q_f, q'_f\rangle)$, where the transition relation $\Delta^\cap$ is the smallest such that for all $q, r \in Q, q', r' \in Q'$, and $\tau \in \{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow\}$,

—$(q, \tau, r) \in \Delta$ and $(q', \tau, r') \in \Delta'$ implies $(\langle q, q'\rangle, \tau, \langle r, r'\rangle) \in \Delta^\cap$;
—$(\langle q, q'\rangle, .[p_{\pi_1,q,r}], \langle r, q'\rangle) \in \Delta^\cap$;
—$(\langle q, q'\rangle, .[p_{\pi_2,q',r'}], \langle q, r'\rangle) \in \Delta^\cap$.

Here, $p_{\pi_1,q,r}$ and $p_{\pi_2,q',r'}$ are new labels that will be bound to $\mathsf{loop}((\pi_1)_{q,r})$ and $\mathsf{loop}((\pi_2)_{q,r})$ in the environment $\rho^\cap$. More precisely, we define $\rho^\cap$ as the concatenation of ρ_1 and ρ_2, preceded by pairs of the form $(p_{\pi_1,q,r}, \mathsf{loop}((\pi_1)_{q,r}))$ and $(p_{\pi_2,q',r'}, \mathsf{loop}((\pi_2)_{q',r'})$ for all $q, r \in Q$ and $q', r' \in Q'$. Recall that we assumed that the sets of atomic labels bound by ρ_1 and ρ_2 are disjoint, so the order of the concatenation does not matter. It is easy to see that $\pi^\cap$ and $\rho^\cap$ satisfy (1)–(3). Using the intuitions provided above, it is also not hard to prove that for all XML trees T, $[\![(\pi^\cap)^{\rho^\cap}]\!]^T_{\mathsf{PExpr}} = [\![\pi_1{}^{\rho_1}]\!]^T_{\mathsf{PExpr}} \cap [\![\pi_2{}^{\rho_2}]\!]^T_{\mathsf{PExpr}}$. □

We now prove the central lemma of this section. Recall from Section 2.3 that the size of a $\mathsf{CoreXPath}(*, \cap)$ node expression φ or path expression α (denoted by $|\varphi|$ and $|\alpha|$) is the number of nodes in the syntax tree of that expression, i.e., it is total number of occurrences of constructors, labels, and atomic path expressions $\downarrow, \uparrow, \rightarrow, \leftarrow$, and ".".

LEMMA 16.

(1) *For every* $\mathsf{CoreXPath}(*, \cap)$ *node expression* φ *there is an equivalent* $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$ *node expression* ψ *with* $|\psi| \leq 2^{5|\varphi|}$.
(2) *For every* $\mathsf{CoreXPath}(*, \cap)$ *path expression* α *there is an equivalent* $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$ *EPA* (π, ρ) *such that*
   (*a*) $|\pi|_S \leq 2^{|\alpha|}$;
   (*b*) $|\pi| \leq 2^{2|\alpha|}$;
   (*c*) $|\rho| \leq 2^{5|\alpha|}$.

PROOF. The two claims are proved by simultaneous induction.

Node Expressions. The base case is trivial since all atomic $\mathsf{CoreXPath}(*, \cap)$ node expressions are also $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ node expressions. The inductive argument for the Boolean connectives $\wedge$ and $\neg$ is straightforward. If φ is of the form $\langle\alpha\rangle$, then the induction hypothesis yields that α is equivalent to a $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$ EPA (π, ρ) satisfying the conditions listed under (2). Let π' be obtained from π by adding additional transitions from the final state to itself

labeled by all possible basic steps in the tree, that is, $\downarrow_1$, $\uparrow_1$, $\leftarrow$, $\rightarrow$. Then, let ψ be the node expression $\mathsf{let}\ \rho\ \mathsf{in}\ \mathsf{loop}(\pi')$. It is not hard to see that ψ is equivalent to φ. Moreover, $|\psi| \leq |\rho| + |\pi'| + 2$ which is bounded by $2^{5|\varphi|}$.

Path Expressions. The base case, that is, for α of the form τ, τ^* or ".", is easy: the corresponding path automaton consists only of two states, connected by a single transition labeled appriopriately. For the regular operations /, $\cup$, and $*$, we apply the usual operations on NFAs. As for the corresponding environments, we may assume without loss of generality that the atomic labels bound in one of the two environments do not occur in the other environment, and vice versa, so that we can concatenate the sequences. It is easy to see that the resulting EPA satisfies all conditions.

Let α be of the form $\alpha_1[\varphi]$. By the induction hypothesis, we can find an EPA (π_1, ρ_1) equivalent to α_1 and a $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$ node expression ψ equivalent to φ. We construct the new path automaton by adding one extra state to π_1, which becomes the new final state, and adding an edge from the old final state to the new one, labeled by $.[p]$, for a fresh p. We construct ρ by prefixing ρ_1 with the pair (p, ψ). It is again easy to show that all requirements are satisfied. In particular,

—$|\pi|_S = |\pi_1|_S + 1 =_{\text{IH}} 2^{|\alpha_1|} + 1 \leq 2^{|\alpha|}$
—$|\pi| = |\pi_1| + 3 =_{\text{IH}} 2^{2|\alpha_1|} + 3 \leq 2^{2|\alpha|}$
—$|\rho| = |\rho_1| + |\psi| + 1 \leq_{\text{IH}} 2^{5|\alpha_1|} + 2^{5|\varphi|} + 1 \leq 2^{5|\alpha|}$

where the subscript IH indicates that we use the induction hypothesis.

Finally, let α be of the form $\alpha_1 \cap \alpha_2$. By the induction hypothesis, there are EPAs (π_1, ρ_1) and (π_2, ρ_2) for α_1 and α_2. Take $(\pi^\cap, \rho^\cap)$ from Lemma 15. We will show that $(\pi^\cap, \rho^\cap)$ satisfies all requirements. In the following equations, we use Lem in subscript to indicate use of the properties listed in Lemma 15, and IH to indicate that we use the induction hypothesis.

$$
\begin{aligned}
|\pi^\cap|_S &=_{\text{Lem}} |\pi_1|_S \cdot |\pi_2|_S \leq_{\text{IH}} 2^{|\alpha_1|} \cdot 2^{|\alpha_2|} \leq 2^{|\alpha|} \\
|\pi^\cap| &=_{\text{Lem}} |\pi_1|_S^2 \cdot |\pi_2|_S + |\pi_2|_S^2 \cdot |\pi_1|_S + |\pi_2|_S \cdot |\pi_1|_S \\
&\leq_{\text{IH}} 3(2^{2|\alpha_1|} \cdot 2^{2|\alpha_2|}) \\
&\leq 2^{2|\alpha|} \\
|\rho^\cap| &\leq_{\text{Lem}} |\rho_1| + |\rho_2| + |\pi_1|_S^2 \cdot (|\pi_1| + 2) + |\pi_2|_S^2 \cdot (|\pi_2| + 2) \\
&\leq_{\text{IH}} 2^{5|\alpha_1|} + 2^{5|\alpha_2|} + 2^{2|\alpha_1|} \cdot (2^{2|\alpha_1|} + 2) + 2^{2|\alpha_2|} \cdot (2^{2|\alpha_2|} + 2) \\
&\leq 2^{5|\alpha|}
\end{aligned}
$$

□

The translation in the proof of Lemma 16 results in an exponential blow-up in the size of expressions. This blow-up disappears if we impose a constant bound on the nesting depth of path intersection, which is defined in the natural way. Formally, we first define the *direct* intersection depth of a path expression: $dd(\tau) = dd(\tau^*) = 0$ for $\tau \in \{\downarrow, \uparrow, \rightarrow, \leftarrow\}$, $dd(\alpha/\beta) = dd(\alpha \cup \beta) = \max\{dd(\alpha), dd(\beta)\}$, $dd(\alpha \cap \beta) = \max\{dd(\alpha), dd(\beta)\} + 1$, and $dd(\alpha[\varphi]) = dd(\alpha)$. Then, the *intersection depth* $d(\alpha)$ of a path expression α (respectively, $d(\varphi)$ of a node expression φ) is the maximum direct intersection depth of any path expression occurring in α (respectively, in φ), possibly inside a node expression.

LEMMA 17. *Let $k \geq 0$ be a constant.*

(1) *For every* CoreXPath$(*, \cap)$ *node expression φ with $d(\varphi) \leq k$, there is an equivalent* CoreXPath$_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$ *node expression ψ with $|\psi| \leq |\varphi|^{2^{k+2}}$.*

(2) *For every* CoreXPath$(*, \cap)$ *path expression α with $d(\alpha) \leq k$, there is an equivalent* CoreXPath$_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$ *EPA (π, ρ) such that*
   (a) $|\pi|_S \leq |\alpha|^{2^k}$;
   (b) $|\pi| \leq |\alpha|^{2^{k+1}}$;
   (c) $|\rho| \leq |\alpha|^{2^{k+2}}$.

PROOF. The translation is identical to the one in the proof of Lemma 16. Here, we only (re)count the size of the obtained expressions and automata. Throughout the calculations, we make use of the binomial theorem, indicating nontrivial applications with "Bin". We only consider the nontrivial cases of the induction.

Node Expressions. Let φ be of the form $\langle\alpha\rangle$, and let ψ be constructed as in the proof of Lemma 16. Then $|\psi| \leq |\rho| + |\pi'| + 2 \leq_{\text{IH}} |\alpha|^{2^{k+2}} + |\alpha|^{2^{k+1}} + 2 \leq_{\text{Bin}} |\varphi|^{2^{k+2}}$.

Path Expressions. Let α be of the form $\alpha_1[\varphi]$. By induction hypothesis, we obtain an EPA (π_1, ρ_1) equivalent to α_1 and a CoreXPath$_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$ node expression ψ equivalent to φ. We construct the new EPA (π, ρ) as in the proof of Lemma 16.

Then, we have:

—$|\pi|_S = |\pi_1|_S + 1 =_{\text{IH}} |\alpha_1|^{2^k} + 1 \leq |\alpha|^{2^k}$
—$|\pi| = |\pi_1| + 3 =_{\text{IH}} |\alpha_1|^{2^{k+1}} + 2 \leq_{\text{Bin}} |\alpha|^{2^{k+1}}$
—$|\rho| = |\rho_1| + |\psi| + 1 \leq_{\text{IH}} |\alpha_1|^{2^{k+2}} + |\varphi|^{2^{k+2}} + 1 \leq_{\text{Bin}} |\alpha|^{2^{k+2}}$

Now let α be of the form $\alpha_1 \cap \alpha_2$. From the induction hypothesis, we obtain EPAs (π_1, ρ_1) and (π_2, ρ_2) for α_1 and α_2. As in the proof of Lemma 16, take $(\pi^\cap, \rho^\cap)$ from Lemma 15. Then

$$\begin{aligned}
|\pi^\cap|_S &=_{\text{Lem}} |\pi_1|_S \cdot |\pi_2|_S \\
&\leq_{\text{IH}} |\alpha_1|^{2^{k-1}} \cdot |\alpha_2|^{2^{k-1}} \\
&\leq |\alpha|^{2^k}
\end{aligned}$$

$$\begin{aligned}
|\pi^\cap| &=_{\text{Lem}} |\pi_1|_S^2 \cdot |\pi_2|_S + |\pi_2|_S^2 \cdot |\pi_1|_S + |\pi_1|_S \cdot |\pi_2|_S \\
&\leq_{\text{IH}} \left(|\alpha_1|^{2^{k-1}}\right)^2 \cdot |\alpha_2|^{2^{k-1}} + \left(|\alpha_2|^{2^{k-1}}\right)^2 \cdot |\alpha_1|^{2^{k-1}} + |\alpha_1|^{2^{k-1}} \cdot |\alpha_2|^{2^{k-1}} \\
&= |\alpha_1|^{2^k} \cdot |\alpha_2|^{2^{k-1}} + |\alpha_2|^{2^k} \cdot |\alpha_1|^{2^{k-1}} + |\alpha_1|^{2^{k-1}} \cdot |\alpha_2|^{2^{k-1}} \\
&\leq_{\text{Bin}} |\alpha|^{2^{k+1}}
\end{aligned}$$

$$\begin{aligned}
|\rho^\cap| &=_{\text{Lem}} |\rho_1| + |\rho_2| + |\pi_1|_S^2 \cdot (|\pi_1| + 2) + |\pi_2|_S^2 \cdot (|\pi_2| + 2) \\
&\leq_{\text{IH}} |\alpha_1|^{2^{k+2}} + |\alpha_2|^{2^{k+2}} + |\alpha_1|^{2^k} \cdot (|\alpha_1|^{2^{k+1}} + 2) + |\alpha_2|^{2^k} \cdot (|\alpha_2|^{2^{k+1}} + 2) \\
&= |\alpha_1|^{2^{k+2}} + |\alpha_2|^{2^{k+2}} + |\alpha_1|^{2^{k+2}} + 2|\alpha_1|^{2^k} + |\alpha_2|^{2^{k+2}} + 2|\alpha_2|^{2^k} \\
&= 2|\alpha_1|^{2^{k+2}} + 2|\alpha_2|^{2^{k+2}} + 2|\alpha_1|^{2^k} + 2|\alpha_2|^{2^k} \\
&\leq_{\text{Bin}} |\alpha|^{2^{k+2}}
\end{aligned}$$

□

4.3. PUTTING THINGS TOGETHER. Lemma 16 provides a single exponential translation from $\mathsf{CoreXPath}(*, \cap)$ to $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$. From Section 3, we also know that satisfiability of $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ node expressions is decidable in EXPTIME. The following lemma allows us to combine these two facts by eliminating the let construct.

LEMMA 18. *Given any* $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$ *node expression* φ *one can compute in polynomial time an equi-satisfiable* $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ *node expression* φ'.

PROOF. The idea is to materialize in XML trees the labels that are bound by let constructs, and to "axiomatize" that they behave in the same way as the formulas they abbreviate. A problem is posed by the fact that nodes in XML trees can only have one label, but different labels from let constructs may abbreviate formulas that are not interpreted disjointly. The solution is to shift such labels from the node where they are supposed to hold to a newly introduced child.

Let φ be a $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop}, \mathsf{let})$ node expression and P the set of labels bound by some occurrence of the let construct in φ. We may assume without loss of generality that (i) no atomic label is bound by two different occurrences of the let construct, and (ii) for any subformula of φ of the form $\mathsf{let}\ p := \psi\ \mathsf{in}\ \chi$, the only (other) occurrences of p in φ are inside χ.

We first introduce some convenient shorthands. For node expressions ψ, χ, and atomic labels p, let

—$\psi_{\downarrow p}$ denote the result of replacing all occurrences of the label p in ψ with $\langle\downarrow[p]\rangle$;
—$\mathsf{equiv}(\psi, \chi)$ be shorthand for $\neg(\langle\uparrow^*/\downarrow^*[\psi \wedge \neg\chi]\rangle \vee \langle\uparrow^*/\downarrow^*[\chi \wedge \neg\psi]\rangle)$, expressing that ψ and χ are true at precisely the same nodes.

For any subnode expression ψ of φ, let ψ^* be the node expression obtained from ψ by replacing

(1) all subexpressions $\mathsf{let}\ p := \theta\ \mathsf{in}\ \chi$ with $\chi_{\downarrow p}$,
(2) every axis $\tau \in \{\downarrow, \uparrow, \leftarrow, \rightarrow\}$ with $\tau[\neg \bigvee_{p\in P}]$, and likewise for the transitive closures of these axes.

Note that replacing all axes τ by $\tau[\neg \bigvee_{p\in P}]$ in this way has the consequence that the obtained expression is blind to nodes satisfying $\bigvee_{p\in P}$, and hence, its truth value is not affected by the introduction of the new nodes.

Finally, let φ' be the conjunction of φ^* and, for every subexpression $\mathsf{let}\ p := \psi\ \mathsf{in}\ \chi$ of φ, the expression

$$\mathsf{equiv}(\langle\downarrow p\rangle, \psi^*) \ \wedge\ \neg\langle\downarrow^*[p]/\downarrow\rangle \ \wedge\ \neg\left\langle\downarrow^*[p]/\rightarrow[\neg \bigvee_{q\in P} q]\right\rangle.$$

The last two conjuncts simply say that the newly introduced nodes are leafs and cannot have "normal" nodes to their right (this is necessary, for example, to preserve unsatisfiability of node expressions $\langle\downarrow^*\rangle \wedge \neg\langle\downarrow\rangle$ and $\langle\rightarrow^*\rangle \wedge \neg\langle\rightarrow\rangle$, respectively). It can be shown that φ' is satisfiable if, and only if, φ is, and that $|\varphi'|$ is quadratic in $|\varphi|$. □

It is clear from the proofs that the size bounds in Lemma 16 and 17 are also bounds on the time needed to compute the translations. Thus, by combining Lemma 16, Lemma 17, and Lemma 18 with Theorem 13, we obtain

THEOREM 19. *Path containment for* CoreXPath($*$, $\cap$) *is in* 2-EXPTIME. *It is in* EXPTIME *when there is a constant bound on the intersection depth of the input expression.*

By Proposition 6, the same holds for path containment relative to an EDTD. A matching lower bound will be proved in Section 6.

5. CoreXPath$_{\downarrow}(\cap)$ *is in ExpSpace*

We consider CoreXPath$_{\downarrow}(\cap)$, the downward fragment of CoreXPath$(\cap)$, and show that path containment can be decided in EXPSPACE. As before, we exploit Proposition 4 and prove the upper bound only for node satisfiability. In contrast to the previous sections, we prove the upper bound directly for the case with EDTDs. By Proposition 5, it carries over to the case without EDTDs. Our proof is inspired by Ladner's PSPACE algorithm for satisfiability in the modal logic K4 [Ladner 1977].

As a preliminary, we show how to get rid of union and intersection in path expressions, at the expense of an exponential blowup. We call a CoreXPath$_{\downarrow}(\cap)$ path expression *simple* if it is of the form $\alpha_1/\cdots/\alpha_n$, $n \geq 1$, where each α_i is of the form $\downarrow$, $\downarrow^*$ or $.[\varphi]$.

LEMMA 20. *For every* CoreXPath$_{\downarrow}(\cap)$ *path expression* α*, there is a set of simple* CoreXPath$_{\downarrow}(\cap)$ *path expressions* $\mathsf{inst}(\alpha)$ *such that*

(i) $|\mathsf{inst}(\alpha)|$ *is* $2^{O(|\alpha|^2)}$,

(ii) *for each* $\beta \in \mathsf{inst}(\alpha)$, $|\beta| \leq 4 \cdot |\alpha|$,

(iii) α *is equivalent to* $\bigcup \mathsf{inst}(\alpha)$, *and*

(iv) *each* $\beta \in \mathsf{inst}(\alpha)$ *contains only node expressions that occur in* α.

PROOF. We proceed in two steps. First, we show that the intersection of two simple path expressions can be written as a union of simple path expressions. To simplify the presentation, during this proof we also admit the simple path expression ε of length zero (i.e., a concatenation of zero steps, which is equivalent to $.[\top]$), and treat α/ε as identical to α. For simple path expressions α and β (which may be ε), we define $\mathsf{int}\{\alpha, \beta\}$ by induction on $|\alpha| + |\beta|$ as follows:

$$
\begin{array}{ll}
\mathsf{int}\{\alpha\} & := \{\alpha\} \\
\mathsf{int}\{\varepsilon, .[\varphi]/\beta\} & := \{.[\varphi]/\gamma \mid \gamma \in \mathsf{int}\{\varepsilon, \beta\}\} \\
\mathsf{int}\{\varepsilon, \downarrow/\beta\} & := \emptyset \\
\mathsf{int}\{\varepsilon, \downarrow^*/\beta\} & := \mathsf{int}\{\varepsilon, \beta\} \\
\mathsf{int}\{.[\varphi]/\alpha, \beta\} & := \{.[\varphi]/\gamma \mid \gamma \in \mathsf{int}\{\alpha, \beta\}\} \\
\mathsf{int}\{\downarrow/\alpha, \downarrow/\beta\} & := \{\downarrow/\gamma \mid \gamma \in \mathsf{int}\{\alpha, \beta\}\} \\
\mathsf{int}\{\downarrow/\alpha, \downarrow^*/\beta\} & := \mathsf{int}\{\downarrow/\alpha, \beta\} \cup \{\downarrow/\gamma \mid \gamma \in \mathsf{int}\{\alpha, \downarrow^*\beta\}\} \\
\mathsf{int}\{\downarrow^*/\alpha, \downarrow^*/\beta\} & := \{\downarrow^*/\gamma \mid \gamma \in \mathsf{int}\{\downarrow^*/\alpha, \beta\}\} \cup \{\downarrow^*/\gamma \mid \gamma \in \mathsf{int}\{\alpha, \downarrow^*/\beta\}\}
\end{array}
$$

By induction on $|\alpha| + |\beta|$, one can show that $\bigcup \mathsf{int}\{\alpha, \beta\}$ is equivalent to $\alpha \cap \beta$, $|\mathsf{int}\{\alpha, \beta\}|$ is bounded by $2^{2(|\alpha|+|\beta|)}$, and each $\gamma \in \mathsf{int}\{\alpha, \beta\}$ satisfies $|\gamma| \leq |\alpha| + |\beta|$.

For any $\mathsf{CoreXPath}_{\downarrow}(\cap)$ path expression α, we now define the set of simple path expressions $\mathsf{inst}(\alpha)$ inductively as follows:

$$\begin{array}{ll}
\mathsf{inst}(\alpha) & := \{\alpha\} \text{ for } \alpha \text{ of the form } \downarrow, \downarrow^*, \text{ or } .[\varphi] \\
\mathsf{inst}(\downarrow[\varphi]) & := \{\downarrow/.[\varphi]\} \\
\mathsf{inst}(\downarrow^*[\varphi]) & := \{\downarrow^*/.[\varphi]\} \\
\mathsf{inst}(.) & := \{.[\top]\} \\
\mathsf{inst}(\alpha_1/\alpha_2) & := \{\alpha_1'/\alpha_2' \mid \text{ each } \alpha_i' \in \mathsf{inst}(\alpha_i)\} \\
\mathsf{inst}(\alpha_1 \cup \alpha_2) & := \mathsf{inst}(\alpha_1) \cup \mathsf{inst}(\alpha_2) \\
\mathsf{inst}(\alpha_1 \cap \alpha_2) & := \bigcup\{\mathsf{int}\{\alpha_1', \alpha_2'\} \mid \text{ each } \alpha_i' \in \mathsf{inst}(\alpha_i)\}
\end{array}$$

By induction on $|\alpha| + |\beta|$, one can show that $\bigcup \mathsf{inst}(\alpha)$ is equivalent to α, $|\mathsf{inst}(\alpha)|$ is bounded by $2^{4(|\alpha|^2)}$, and each $\gamma \in \mathsf{inst}(\alpha)$ satisfies $|\gamma| \leq 4 \cdot |\alpha|$ (note that $|\mathsf{inst}(\alpha_1 \cap \alpha_2)| \leq 2^{8|\alpha_1|+8|\alpha_2|} \cdot 2^{4|\alpha_1|^2} \cdot 2^{4|\alpha_2|^2}$, which, by the Binomial theorem, is less than $2^{4(|\alpha_1|+|\alpha_2|+1)} = 2^{4|\alpha_1 \cap \alpha_2|}$). Moreover, all node expressions occurring in expressions from $\mathsf{inst}(\alpha)$ occur in α. □

We can now define, for each $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression φ, two important sets of expressions, namely $\mathsf{sub}(\varphi)$ and $\mathsf{aux}(\varphi)$. The set $\mathsf{sub}(\varphi)$ consists of all node subexpressions of φ, that is, node expressions that occur as a subterm of φ (possibly nested inside a path expression occurring in φ). The set $\mathsf{aux}(\varphi)$ consists of all node expressions of the form $\langle\beta\rangle$ where β is a suffix of a simple path expression in $\mathsf{inst}(\alpha)$ for some $\langle\alpha\rangle \in \mathsf{sub}(\varphi)$. Here, by a *suffix* of a simple path expression $\alpha_1/\cdots/\alpha_n$, we will mean any simple path expression $\alpha_i/\cdots/\alpha_n$ $(1 \leq i \leq n)$.

To illustrate these definitions, consider the $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression

$$\varphi \;=\; p \wedge \langle\downarrow^*[q]/\downarrow^* \cap \downarrow^*[r]/\downarrow^*\rangle$$

The set of subexpressions $\mathsf{sub}(\varphi)$ consists only of φ itself, p, $\langle\downarrow^*[q]/\downarrow^* \cap \downarrow^*[r]/\downarrow^*\rangle$, q, and r. As the reader may verify, $\mathsf{inst}(\downarrow^*[q]/\downarrow^* \cap \downarrow^*[r]/\downarrow^*) = \{\downarrow^*/.[q]/\downarrow^*/.[r]/\downarrow^*, \downarrow^*/.[q]/\downarrow^*/.[r], \downarrow^*/.[r]/\downarrow^*/.[q]/\downarrow^*, \downarrow^*/.[r]/\downarrow^*/.[q]\}$, and therefore $\mathsf{aux}(\varphi)$ consists of all node expressions $\langle\beta\rangle$ with β a suffix of one of these four simple path expressions.

An important intuition to keep in mind is that $\mathsf{sub}(\varphi) \cup \mathsf{aux}(\varphi)$ contains all node expressions that one might need to consider in order to compute the set of nodes satisfying φ by induction on node expressions only (as opposed to a simultaneous induction on node expressions and path expressions).

LEMMA 21. *If a* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *node expression* φ_0 *is satisfiable with respect to an EDTD* $D = (\Delta, P, r, \mu)$*, then it is satisfiable in an XML tree of height* $2^{O(|\varphi_0|^3)} \cdot |\Delta|$ *conforming to* D.

PROOF. The basic idea of the proof, which is essentially a pumping argument, is that every sufficiently long branch in an XML tree satisfying φ_0 must contain two nodes, say $nR_{\downarrow^+}m$, that are of the same "type" (recall that we use $\downarrow^+$ as a shorthand for $\downarrow/\downarrow^*$, that is, $R_{\downarrow^+}$ is the proper descendant relation). Then, we can shorten the path by replacing the subtree rooted at n with the subtree rooted at m, while preserving the truth of φ_0. The main difficulty is to formulate the right notion of a "type" of a node n. Roughly speaking, it contains the following information: (i) it specifies which subformulas of φ_0 are satisfied at n, as well as at each ancestor of n within a certain small distance, (ii) whenever n (or one of the mentioned ancestors) satisfies a subexpression of φ_0 of the form $\langle\alpha\rangle$, the type

function indicates a $\beta \in \mathsf{inst}(\alpha)$ with $\langle\beta\rangle$ satisfied at n and a witnessing β-path; and (iii) it specifies the abstract label that is assigned to n by the function L' witnessing that the tree conforms to the EDTD (c.f., Definition 2).

For the remaining proof, let φ_0 and $D = (\Delta, P, r, \mu)$ be as in Lemma 21, and assume that Δ is disjoint from the image of μ. Moreover, fix an XML tree T that contains a node satisfying φ_0 and conforms to D. We first define a function π that assigns to each node n a set of node expressions from $\mathsf{aux}(\varphi_0)$ that are true at n, such that the following conditions hold for all nodes n:

(1) If n satisfies some $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$, then $\pi(n)$ includes a formula $\langle\beta\rangle$ with $\beta \in \mathsf{inst}(\alpha)$.

(2) For all $\langle.[\varphi]/\beta\rangle \in \pi(n)$, also $\langle\beta\rangle \in \pi(n)$.

(3) For all $\langle\beta/\beta'\rangle \in \pi(n)$ with $\beta \in \{\downarrow, \downarrow^*\}$, there is a node n' such that $nR_\beta n'$ and $\langle\beta'\rangle \in \pi(n')$.

Observe that π fixes witnessing paths in the sense of comment (ii) above, hence we call π a witness function. The construction of π proceeds as follows. Start with the function π_1 that assigns to each node n a set containing, for each $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$, a formula $\langle\beta\rangle$ with $\beta \in \mathsf{inst}(\alpha)$, in order to satisfy condition (1) above. Next, for each $i \geq 1$, if π_i violates condition (2) or (3), extend π_i to π_{i+1} by fixing all immediate violations of (2) and (3) at the same time. Specifically, whenever $\langle.[\varphi]/\beta\rangle \in \pi_i(n)$ and $\langle\beta\rangle \notin \pi_i(n)$, we add $\langle\beta\rangle$ to $\pi_{i+1}(n)$. And whenever $\langle\beta/\beta'\rangle \in \pi_i(n)$ with $\beta \in \{\downarrow, \downarrow^*\}$ and there is no node n' such that $nR_\beta n'$ and $\langle\beta'\rangle \in \pi(n')$, then choose a node n' with $nR_\beta n'$ and $n' \in [\![\langle\beta'\rangle]\!]^T_{\mathsf{NExpr}}$ and add $\langle\beta'\rangle$ to $\pi(n')$. It is not hard to see that, after a finite number of steps, a function π_i will be obtained that satisfies all requirements of a witness function. Indeed, since at each step the expressions that are added get shorter, the total number of steps needed is bounded by the maximum length of the expressions $\beta \in \mathsf{inst}(\alpha)$ with $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$, which, by Lemma 20, is at most $4|\varphi_0|$.

A crucial property of the witness function π constructed in this way is

(∗) On every root-to-leaf branch $n_0, \ldots, n_k$ of T, $|\pi(n_0)| + \cdots + |\pi(n_k)|$ is $O(k \cdot |\varphi_0|^2)$.

This can be shown by induction on the number of steps in the construction of the witness function π. More precisely, a straightforward induction argument shows that, for each $i \geq 1$, π_i consists of at most $i \cdot k \cdot |\varphi_0|$ many assignments of expressions to nodes, of which at most $k \cdot |\varphi_0|$ many are not already in π_{i-1}. Since $\pi = \pi_i$ for some $i \leq 4|\varphi_0|$, it follows that (∗) is satisfied.

Fix a mapping L' that satisfies Conditions (i) to (iii) from Definition 2. For any node n and integer $k \geq 0$, let $n-k$ denote the ancestor of n at distance k (thus, $n-0$ is n, $n-1$ is the parent of n, etc.). Let $\mathsf{aux}_{\downarrow^*}(\varphi)$ be the subset of $\mathsf{aux}(\varphi)$ containing those $\langle\alpha\rangle \in \mathsf{aux}(\varphi)$ that are of the form $\langle\downarrow^*/\alpha'\rangle$. For any node n, we define $\mathsf{type}(n)$ to be the following set:

$$\begin{aligned}\mathsf{type}(n) = \; & \{(k, \psi) \in \{0, \ldots, d\} \times (\mathsf{sub}(\varphi_0) \cup \mathsf{aux}_{\downarrow^*}(\varphi_0)) \mid \\ & n - k \text{ exists and satisfies } \psi\} \\ & \cup \pi(n) \cup \{L'(n)\}.\end{aligned}$$

Note that $(0, \psi) \in \mathsf{type}(n)$ if, and only if, n satisfies ψ, for $\psi \in \mathsf{sub}(\varphi_0) \cup \mathsf{aux}_{\downarrow^*}(\varphi_0)$.

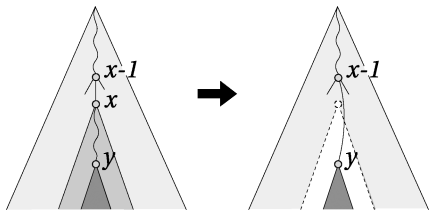
CLAIM 1. *On each branch of T, at most $2^{O(|\varphi_0|^3)}$ distinct types can be realized.*

PROOF OF CLAIM. Since $|\mathsf{sub}(\varphi_0)| \leq |\varphi_0|$, there are at most $2^{|\varphi_0|}$ many distinct subsets of $\mathsf{sub}(\varphi_0)$ to be realized in T. While $|\mathsf{aux}_{\downarrow^*}(\varphi_0)|$ is $2^{O(|\varphi_0|^2)}$ (cf. Lemma 20), on each single branch in T at most $|\mathsf{aux}_{\downarrow^*}(\varphi_0)|$ many subsets of $\mathsf{aux}_{\downarrow^*}(\varphi_0)$ can be realized. This is due to the volatile nature of these expressions: if an expression of the form $\langle\downarrow^*/\cdots\rangle$ is false at a node n, it remains false at all descendants of n. Thus, at most $2^{O(\varphi_0)} \cdot 2^{O(|\varphi_0|^2)} = 2^{O(|\varphi_0|^2)}$ subsets of $\mathsf{sub}(\varphi_0) \times \mathsf{aux}(\varphi_0)$ can be realized on each branch. However, since the type of a node also specifies which formulas in $\mathsf{sub}(\varphi_0) \cup \mathsf{aux}(\varphi_0)$ are satisfied by the kth *ancestor*, for all $k \leq d$, this number should further be raised to the power $O(|\varphi_0|)$, giving us a bound, so far, of $2^{O(|\varphi_0|^3)}$. □

Next, consider the node expressions assigned to the nodes by the witness function π. By $(*)$, the average $|\pi(n)|$ is at most $O(|\varphi_0|^2)$. It follows by basic combinatorics that there can be no more than $2^{O(|\varphi_0|^2)}$ many nodes with distinct π-values: the relevant combinatorial fact is that there is no family of j^{i+1} distinct subsets of $\{1, 2, \ldots, j\}$ of average cardinality less than i (where the set $\{1, 2, \ldots, j\}$ corresponds to the set of pairs (n, ψ) with n a node on the branch and $\psi \in \pi(n)$).

Finally, every type contains precisely one element of Δ, which additionally gives $|\Delta|$ choices. Multiplying the numbers, we obtain that at most $2^{O(|\varphi_0|^3)} \cdot |\Delta|$ many types can be realized on any branch. END OF PROOF OF CLAIM.

CLAIM 2. *If T satisfies φ_0 at the root and contains distinct nodes $x R_{\downarrow^*} y$ that have the same type, then the following contraction preserves truth of φ_0 at the root:*



"remove the subtree rooted at x, and replace it with the subtree rooted at y"

PROOF OF CLAIM. We prove that in the contracted XML tree, all remaining nodes satisfy the same node expressions $\psi \in \mathsf{sub}(\varphi_0)$ as in the original XML tree. The proof is by induction on ψ. The only difficult case is where ψ is of the form $\langle\alpha\rangle$. Let n be any node in the contracted XML tree.

[$\Rightarrow$] Suppose n satisfies $\langle\alpha\rangle$ in the original XML tree. Then there must be a $\beta = (\beta_1/\cdots/\beta_k) \in \mathsf{inst}(\alpha)$ such that $\langle\beta\rangle \in \pi(n)$. We will show by induction on k that n still satisfies $\langle\beta\rangle$, and hence $\langle\alpha\rangle$, in the contracted XML tree. (Notice that we are doing a double induction: the outer induction is on the complexity of ψ and the inner induction is on k.) To reduce the number of cases in the induction argument, it will be convenient to consider also the empty composition ε (i.e., the composition of zero path expressions) as a simple path expression defining the identity relation. The base case of the (inner) induction is then the case where $k = 0$ and thus $\beta = \varepsilon$, and it is trivial.

For the inductive step, we distinguish three cases:

—$\beta_1 = .[\psi]$ for some $\psi \in \mathsf{sub}(\varphi_0)$. In this case, by the definition of π, n satisfies ψ in the original XML tree, and $\langle \beta_2/\cdots/\beta_k\rangle \in \pi(n)$. It follows by the induction hypothesis, that n also satisfies ψ and $\langle \beta_2/\cdots/\beta_k\rangle$, and hence also $\langle\beta\rangle$, in the contracted XML tree.

—$\beta_1 = \downarrow$. Then, by the definition of π, there must be a child n' of n such that $\langle \beta_2/\cdots/\beta_k\rangle \in \pi(n')$. If $n' \neq x$ then it follows from the induction hypothesis that n' satisfies $\langle \beta_2/\cdots/\beta_k\rangle$ in the contracted XML tree, and hence n satisfies $\langle\beta\rangle$ in the contracted XML tree. If, on the other hand, $n' = x$, then, as x and y have the same type, $\langle \beta_2/\cdots/\beta_k\rangle$ must be satisfied by y in the original XML tree. It follows by the induction hypothesis that y satisfies $\langle \beta_2/\cdots/\beta_k\rangle$ in the contracted XML tree, and therefore n satisfies $\langle\beta\rangle$ in that XML tree.

—$\beta_1 = \downarrow^*$. Then by the definition of π, there must be a descendant n' of n such that $\langle \beta_2/\cdots/\beta_k\rangle \in \pi(n')$. If n' belongs to the contracted XML tree, then we can infer from the induction hypothesis that n' satisfies $\langle \beta_2/\cdots/\beta_k\rangle$ in the contracted XML tree, and hence n satisfies $\langle\beta\rangle$ in the contracted XML tree. If, on the other hand, n' is removed during the contraction, then it must be a descendant of x, which implies that x satisfies $\langle\beta\rangle$. Since $\langle\beta\rangle \in \mathsf{aux}_{\downarrow^*}(\varphi_0)$, it follows that y also satisfies $\langle\beta\rangle$, and hence there is a descendant of n'' of y satisfying $\langle \beta_2/\cdots/\beta_k\rangle$. Since the subXML tree rooted at y is not affected by the contraction, y must still satisfy $\langle \beta_2/\cdots/\beta_k\rangle$ in the contracted XML tree. We also know that n'' is a descendant of n, and hence n satisfies $\langle\beta\rangle$ in the contracted XML tree.

[$\Leftarrow$] Suppose n satisfies $\langle\alpha\rangle$ in the contracted XML tree. Then there must be a $\beta \in \mathsf{inst}(\alpha)$ such that n satisfies $\langle\beta\rangle$ in the contracted XML tree. We show that n satisfies $\langle\beta\rangle$ in the original XML tree, again by induction on the number of steps in β. We distinguish two cases: if β is of the form $(\downarrow^*/\vec{\gamma})$, then some descendant n' of n satisfies $\langle\vec{\gamma}\rangle$ in the contracted XML tree. But then n' is also a descendant of n in the original XML tree, and by induction hypothesis, it satisfies $\langle\vec{\gamma}\rangle$. Therefore, n satisfies $\langle\beta\rangle$ in the original XML tree.

It remains to consider the case where β does not start with a $\downarrow^*$ step. Let $\beta = (\beta_1/\cdots/\beta_k/\vec{\gamma})$, with $k \geq 1$, such that each β_i is of the form $\downarrow$ or $.[\psi]$, and $\vec{\gamma}$ is either empty or starts with $\downarrow^*$. Let $n_1, \ldots, n_k$ be nodes in the contracted XML tree such that $n_1 = n$, (n_i, n_{i+1}) belongs to β_i in the contracted XML tree, for $1 \leq i < k$, and n_k satisfies $\langle\vec{\gamma}\rangle$. We distinguish two cases:

—Either $n_i \neq y$ for all $1 \leq i \leq k$, or $n_1 = y$. In this case, the same path $n_1, \ldots, n_k$ exists also in the original XML tree. Moreover, whenever β_i is of the form $.[\psi]$, then, by induction hypothesis, n_i satisfies ψ in the original XML tree as well, and, for the same reason, n_k satisfies $\langle\vec{\gamma}\rangle$. Thus, $\langle\beta\rangle$ must be true at n in the original XML tree.

—$n_i = y$ for some i with $1 \leq i \leq k$, and $n_1 \neq y$. It follows that the sequence $n_1, \ldots, n_k$ is of the form $x - \ell, \ldots, x - 1, y, \ldots, y + m$, where $\ell \leq d$. Define a new sequence $n'_0, \ldots, n'_k$ by replacing each $x - i$ by $y - i$. Thus, the resulting sequence $n'_1, \ldots, n'_k$ is of the form $y - \ell, \ldots, y - 1, y, \ldots, y + m$. Since x and y have the same type in the original XML tree, n_i and n'_i agree on all node expressions from $\mathsf{sub}(\varphi_0)$ (in the original XML tree). Moreover, whenever β_i is of the form $.[\psi]$, then, by induction hypothesis, we know that n_i satisfies ψ in

the original XML tree as well. Together this implies that $(y - \ell, y + m)$ satisfies $(\beta_1/\cdots/\beta_k)$ in the original XML tree. Finally, by (inner) induction hypothesis, $n'_k = n_k$ satisfies $\langle\vec{\gamma}\rangle$ in the original XML tree. We conclude that n'_1 satisfies $\langle\beta\rangle$ and hence $\langle\alpha\rangle$ in the original XML tree. Finally, using once more the fact that x and y have the same type in the original XML tree, we conclude that n_1 must also satisfy $\langle\alpha\rangle$ in the original XML tree (recall that $n_1 = x - \ell$, $n'_1 = y - \ell$ and $\ell \leq d$). END OF PROOF OF CLAIM.

Finally, we establish Lemma 21. Recall that $T = (N, R_\downarrow, R_\rightarrow, L)$ is an XML tree that satisfies φ_0 and conforms to D, and $L' : N \to \Delta$ a function that satisfies Conditions (i) to (iii) of Definition 2. If T contains a branch of length greater than $2^{p(|\varphi_0|)} \cdot |\Delta|$, with p the polynomial from Claim 1, we can apply the contraction operation from Claim 2 to obtain an XML tree T' with fewer nodes than T that still satisfies φ_0. Let L'' be the restriction of L' to the nodes in T'. Using the fact that $L(n)$ occurs in the type of n for all $n \in N$, it is easy to show that L'' still satisfies Conditions (i) to (iii) of Definition 2. Thus, T' conforms to D. Repeating this contraction argument, we eventually get an XML tree of depth $2^{O(|\varphi_0|^3)} \cdot |\Delta|$. □

To present the decision procedure, we need a few further preliminaries. For any $\mathsf{CoreXPath}_\downarrow(\cap)$ node expression φ, let $\mathsf{cl}(\varphi) = \{\psi, \neg\psi \mid \psi \in \mathsf{sub}(\varphi) \cup \mathsf{aux}(\varphi)\}$. Intuitively, this definition of $\mathsf{cl}(\varphi)$ is the $\mathsf{CoreXPath}_\downarrow(\cap)$ analogue of the definition of $\mathsf{cl}(\varphi)$ for $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ node expressions φ given in Section 3.3. Note that the size of $\mathsf{cl}(\varphi)$ is in general exponential in the length of φ, while the length of the members of $\mathsf{cl}(\varphi)$ is polynomial.

Definition 22 (*Complete Type*). Let φ_0 be a $\mathsf{CoreXPath}_\downarrow(\cap)$ node expression and $D = (\Delta, P, r, \mu)$ an EDTD. A *complete type for* φ_0 *and* D is a set $t \subseteq \mathsf{cl}(\varphi_0) \cup \Delta$ satisfying the following conditions:

—t contains exactly one $s \in \Delta$ and exactly one $p \in \Sigma$ such that $\mu(s) = p$.
—For all $(\varphi \wedge \psi) \in \mathsf{cl}(\varphi_0)$, $(\varphi \wedge \psi) \in t$ if, and only if, $\varphi \in t$ and $\psi \in t$.
—For all $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$, $\langle\alpha\rangle \in t$ if, and only if, there is a $\beta \in \mathsf{inst}(\alpha)$ such that $\langle\beta\rangle \in t$.
—For all $\langle.[\psi]/\beta\rangle \in \mathsf{aux}(\varphi_0)$, $\langle.[\psi]/\beta\rangle \in t$ if, and only if, $\psi \in t$ and $\langle\beta\rangle \in t$.
—For all $\langle\downarrow^*/\beta\rangle \in \mathsf{aux}(\varphi_0)$, if $\langle\beta\rangle \in t$ then $\langle\downarrow^*/\beta\rangle \in t$.

Let t be a complete type for φ_0 and D. A formula $\psi \in t$ is called a *demand* in t if either (i) ψ is of the form $\langle\downarrow/\alpha\rangle$, or (ii) ψ is of the form $\langle\downarrow^*/\alpha\rangle$ and t does not contain $\langle\alpha\rangle$. In the first case, the *remainder* of ψ is $\langle\alpha\rangle$, and in the second case, the remainder of ψ is ψ itself.

Given two complete types t, t' for φ_0 and D, we write $t \Rightarrow t'$ if the following two conditions hold:

—for all $\langle\downarrow/\alpha\rangle \in \mathsf{aux}(\varphi_0)$, if $\langle\alpha\rangle \in t'$ then $\langle\downarrow/\alpha\rangle \in t$;
—for all $\langle\downarrow^*/\alpha\rangle \in \mathsf{aux}(\varphi_0)$, if $\langle\downarrow^*/\alpha\rangle \in t'$ then $\langle\downarrow^*/\alpha\rangle \in t$

Intuitively, $t \Rightarrow t'$ means that we can consistently think of t' as being the complete type of a child of a node with complete type t.

The (nondeterministic) algorithm for node satisfiability in $\mathsf{CoreXPath}_\downarrow(\cap)$ is given in Figure 2. It takes as input a node expression φ_0 and an EDTD $D = (\Delta, P, r, \mu)$, and it tries to recursively construct an XML tree satisfying the expression that

```
guess a complete type t for φ0 and D = (Δ, P, r, μ);
if φ0 ∉ t or r ∉ t then return 'No' else return check(t,0);

procedure check(t, d):
  if d > 2^{p(|φ0|)} · |Δ| then return 'No' else continue;

  guess complete types t1, ..., tk for φ0 and D with k ≤ (|aux(φ0)| + 1) · |D|
  let s be the (unique) element of Δ contained in t
      and si the (unique) element of Δ contained in ti, for 1 ≤ i ≤ k
  if s1 ··· sk ∉ L(P(s)) then return 'No' else continue;

  for each demand φ ∈ t do
    let the remainder of φ be φ';
    if φ' ∉ t1 ∪ ··· ∪ tk then return 'No' else continue;

  for 1 ≤ i ≤ k do
    if t ⇏ ti or check(ti, d + 1)='No' then return 'No' else continue;

  return 'Yes';
```

FIG. 2. Algorithm for satisfiability of a $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression φ_0 with restect to an EDTD $D = (\Delta, P, r, \mu)$.

conforms to D, while keeping in memory at any point only a single branch of length at most $2^{p(|\varphi_0|)} \cdot |\Delta|$, with p the polynomial from Lemma 21. In the figure, $|D|$ is defined as the maximum number of states of any NFA obtained by converting a regular expression $P(s)$, for some $s \in \Delta$.

THEOREM 23. *The algorithm has an accepting run on input node expression φ_0 and EDTD D if, and only if, φ_0 is satisfiable with respect to D.*

PROOF. [$\Rightarrow$] Suppose the algorithm has an accepting run on input φ_0. Let $T = (N, R_{\downarrow}, R_{\rightarrow}, \tau)$ be the recursion tree of this run, that is, $(N, R_{\downarrow}, R_{\rightarrow})$ is a finite sibling ordered tree, and τ assigns to each node the complete type that is the first argument of the corresponding recursive call of `check`. Take the XML tree $T = (N, R_{\downarrow}, R_{\rightarrow}, L)$, where $L : N \rightarrow \Sigma$ assigns to each node $n \in N$ the unique label from Σ that belongs to $\tau(n)$.

It can be proved that for all $n \in N$ and $\psi \in \mathsf{cl}(\varphi_0)$, $n \in [\![\psi]\!]^T_{\mathsf{NExpr}}$ if, and only if, $\psi \in \tau(n)$. In particular, since $\varphi_0 \in \tau(n_r)$, for n_r the root of T, we have that $n_r \in [\![\varphi_0]\!]^T_{\mathsf{NExpr}}$. The proof is by induction on the well-founded ordering $\prec$ on $\mathsf{cl}(\varphi_0)$ generated by:

—$\varphi \prec \psi$ whenever $\varphi \in \mathsf{sub}(\psi)$;

—$\langle\beta\rangle \prec \langle\alpha\rangle$ whenever $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$, $\langle\beta\rangle \in \mathsf{aux}(\varphi_0)$, and $\beta \in \mathsf{inst}(\alpha)$;

—$\langle\beta'\rangle \prec \langle\beta/\beta'\rangle$ for all $\langle\beta/\beta'\rangle \in \mathsf{aux}(\psi)$

We leave details to the reader. Define $L' : N \rightarrow \Delta$ by setting $L'(n)$ to the unique element of Δ occurring in $\tau(n)$. It is easy to see that L' satisfies Conditions (i) to (iii) from Definition 2, and thus T conforms to D.

[$\Leftarrow$] Suppose that φ_0 is satisfiable with respect to $D = (\Delta, P, r, \mu)$. By Lemma 21, φ_0 is satisfied at the root of an XML tree $T = (N, R_{\downarrow}, R_{\rightarrow}, L)$ of

depth at most $2^{p(|\varphi|)} \cdot |\Delta|$ that conforms to D. We first describe how to eliminate nodes from T to achieve an out-degree of at most $b := (|\mathsf{aux}(\varphi_0)| + 1) \cdot |D|$.

Let $L' : N \to \Delta$ satisfy Conditions (i) to (iii) from Definition 2. We repeat the following for all nodes $n \in N$ with $k > b$ successors $n_1, \ldots, n_k$, in a top-down fashion. Select

(i) an $n_i \in [\![\langle\beta\rangle]\!]^T_{\mathsf{NExpr}}$ for each $\psi = \langle\downarrow/\beta\rangle \in \mathsf{aux}(\varphi_0)$ with $n \in [\![\psi]\!]^T_{\mathsf{NExpr}}$;

(ii) an $n_i \in [\![\psi]\!]^T_{\mathsf{NExpr}}$ for each $\psi = \langle\downarrow^*/\beta\rangle \in \mathsf{aux}(\varphi_0)$ with $n \in [\![\psi]\!]^T_{\mathsf{NExpr}} \setminus [\![\langle\beta\rangle]\!]^T_{\mathsf{NExpr}}$.

Let A be the NFA obtained from converting the regular expression $P(L'(n))$, and let ρ be an accepting run of A on the word $L'(n_1) \cdots L'(n_k)$, that is, ρ is a mapping from $\{1, \ldots, k\}$ to states of A. Then, we can repeat the following elimination step: if there are n_i, n_j with $1 < i < j \leq k$ such that $\rho(i) = \rho(j)$ and none of the nodes $n_i, n_{i+1}, \ldots, n_{j-1}$ is selected, then drop the subtrees rooted at $n_i, \ldots, n_{j-1}$ from T. Following this strategy, we get a trimmed-down version of T with branching factor at most b that still satisfies φ_0 and still conforms to D.

We use the resulting tree T to guide the algorithm. For any node n of T, let

$$\tau(n) = \left\{\varphi \in \mathsf{cl}(\varphi_0) \mid n \in [\![\varphi]\!]^T_{\mathsf{NExpr}}\right\}.$$

To show that the algorithm has an accepting run on input φ_0, we associate to each call of $\mathtt{check}(t, d)$ a "witnessing" node n in the tree, at distance d from the root, satisfying $\tau(n) = t$. Initially, we let the algorithm guess the complete type $\tau(n_r)$ of the root n_r, and n_r is the witnessing node. Next, suppose that a recursive call $\mathtt{check}(t, d)$ is made, and let n be a witnessing node. Let $n_1, \ldots, n_k$ be the (ordered) successors of n. We let the algorithm guess the sequence of complete types $\tau(n_1), \ldots, \tau(n_k)$. Since $\tau(n) = t$ and by the semantics, the algorithm does not return in any step of the first for loop. If $\mathtt{check}(t_i, d + 1)$ is called in the second for loop, we use n_i as the witnessing node. It is easy to see that the algorithm returns "Yes" on input φ_0 and D when all nondeterministic choices are made according to this strategy. □

The above algorithm runs in nondeterministic exponential space. To see this, observe that the recursion depth is bounded by a function that is single exponential in the input expression, and each recursive call adds on the recursion stack a sequence of complete types $t_1, \ldots, t_k$ where k is bounded by a function that is single exponential in the input expression. Finally, each complete type is a subset of $\mathsf{cl}(\varphi)$ and can thus be represented using single exponentially many bits.

Since NEXPSPACE = EXPSPACE by Savitch's theorem, we obtain that node satisfiability for $\mathsf{CoreXPath}_{\downarrow}(\cap)$ with respect to EDTDs is in EXPSPACE. By Proposition 5, this result carries over to the case without EDTDs.

THEOREM 24. *Path containment for* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *is in* EXPSPACE*, with and without EDTDs.*

A matching lower bound is proved in the next section, where we also establish a 2-EXPTIME lower bound for $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$. One may be tempted to think that the latter bound carries over to $\mathsf{CoreXPath}_{\downarrow}(\cap)$ via the standard encoding of trees with arbitrary branching factor into binary trees (cf. the proof of Theorem 10). However, this is not the case since $\mathsf{CoreXPath}_{\downarrow}(\cap)$ does not provide full transitive closure.

6. *Lower Bounds*

We provide matching lower bounds for the upper bounds from the previous sections: path containment is EXPSPACE-hard for $\mathsf{CoreXPath}_{\downarrow}(\cap)$, and it is 2-EXPTIME-hard for $\mathsf{CoreXPath}(\cap, *)$. In fact, we show that the 2-EXPTIME lower bound holds for the fragments $\mathsf{CoreXPath}_{\downarrow\uparrow}(\cap)$, $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$ and $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$. This shows that the EXPSPACE upper bound for $\mathsf{CoreXPath}_{\downarrow}(\cap)$ cannot be generalized in an easy way.

6.1. PRELIMINARIES. We prove our results for satisfiability of node expressions. It is convenient to work with a generalization of XML trees where nodes can satisfy more than one label, that is, where the labeling function L is a function from N to 2^{Σ}. We call such trees *XML trees with multi-labels*. The following lemma shows that, in proving our lower bounds, we can safely use XML trees with multi-labels.

LEMMA 25. *Satisfiability of* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *node expressions on XML trees with multi-labels can be reduced in polynomial time to satisfiability of* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *node expressions on standard XML trees. The same holds for* $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$, $\mathsf{CoreXPath}_{\downarrow\uparrow}(\cap)$ *and* $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$.

PROOF SKETCH. We can turn a multi-labeled XML tree into a standard XML tree by adding $|L(n)|$ extra children to each node n, each of them labeled with a different element of $L(n)$. To distinguish these auxiliary nodes from "real" document nodes, we label the latter with a special node label x. Guided by this idea of encoding trees with multi-labels, it is not hard to transform a $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression φ into a $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression φ' such that φ is satisfiable in an XML tree with multi-labels if, and only if, φ' is satisfiable in a standard XML tree: let φ^* be obtained from φ by (i) replacing every occurrence of $\downarrow$ and $\downarrow^*$ by $\downarrow[x]$ and $\downarrow^*[x]$, and (ii) replacing every occurrence of a node label p with $\langle\downarrow[p]\rangle$. Then, set $\varphi' := \varphi^* \wedge x \wedge \neg\langle\downarrow^*[\neg x]/\downarrow\rangle$. The last conjunct of φ' ensures that all auxiliary nodes are leafs, which is necessary for the same reason as in the (similar) proof of Lemma 18. Similar reductions can be given for $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$, $\mathsf{CoreXPath}_{\downarrow\uparrow}(\cap)$ and $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$. □

For all lower bounds established in this section, we work with XML trees with multi-labels. To start, Lemma 25 allows us to easily derive one of the announced lower bounds because $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$ with multi-labels can be seen as a notational variant of *propositional dynamic logic with intersection* (*IPDL*), restricted to a single atomic program a. For example, the $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$ node expression $\langle\downarrow[a]^*/(\downarrow[b]\cup\downarrow[b'])\rangle$ corresponds to the IPDL formula $\langle(r; a?)^*; (r; b?\cup r; b'?)\rangle\mathsf{true}$ and the IPDL formula $\langle(r; a?)^* \cap (r; b?)^*\rangle c$ corresponds to the $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$ node expression $\langle(\downarrow[a]^* \cap \downarrow[b]^*)[c]\rangle$. It was proved in Lange and Lutz [2005] that satisfiability of ICPDL formulas in finite XML trees (with multi-labels) is 2-EXPTIME-hard.

THEOREM 26. *Path containment for* $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$ *is* 2-EXPTIME-*hard*.

The remaining bounds are proved by reduction of the word problem of alternating Turing machines (ATMs), which we introduce next. An ATM is of the form $\mathcal{M} = (Q, \Lambda, \Gamma, q_0, \Delta)$, where $Q = Q_{\exists} \uplus Q_{\forall} \uplus \{q_a\} \uplus \{q_r\}$ is the set of *states*, partitioned into *existential states* from $Q_{\exists}$, *universal states* from $Q_{\forall}$, an *accepting state* q_a, and a *rejecting state* q_r; Λ is the *input alphabet* and $\Gamma \supseteq \Lambda$ the *work alphabet*

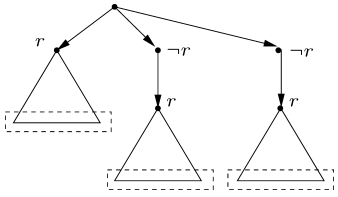


FIG. 3. Successor configurations in $\mathsf{CoreXPath}_{\downarrow,\uparrow}(\cap)$.

containing a *blank symbol* $\Box$; $q_0 \in Q_\exists \cup Q_\forall$ is the *starting* state; and the *transition relation* Δ is of the form $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$. We write $\Delta(q, a)$ for $\{(q', b, M) \mid (q, a, q', b, M) \in \Delta\}$.

A *configuration* of an ATM is a word wqw' with $w, w' \in \Gamma^*$ and $q \in Q$. The intended meaning is that the (one-side infinite) tape contains the word ww' with only blanks behind it, the machine is in state q, and the head is on the leftmost symbol of w'. The *successor configurations* of a configuration wqw' are defined in the usual way in terms of the transition relation Δ. A *halting configuration* is of the form wqw' with $q \in \{q_a, q_r\}$. A *computation* of an ATM $\mathcal{M}$ on a word w is a (finite or infinite) sequence of successive configurations $K_1, K_2, \ldots$. The ATMs considered in the following have only finite computations on any input. Since this case is simpler than the general one, we define acceptance for ATMs with finite computations and refer to Chandra et al. [1981] for the full definition. Let $\mathcal{M}$ be such an ATM. A halting configuration is *accepting* if it is of the form wq_aw'. For other configurations $K = wqw'$, the acceptance behavior depends on q: if $q \in Q_\exists$, then K is accepting if at least one successor configuration is accepting; if $q \in Q_\forall$, then K is accepting if all successor configurations are accepting. Finally, the ATM $\mathcal{M}$ with starting state q_0 *accepts* the input w if the *initial configuration* q_0w is accepting. We use $L(\mathcal{M})$ to denote the language accepted by $\mathcal{M}$, that is, $L(\mathcal{M}) = \{w \in \Lambda^* \mid \mathcal{M} \text{ accepts } w\}$.

6.2. $\mathsf{CoreXPath}_{\downarrow,\uparrow}(\cap)$ IS 2-EXPTIME-hard. There exists an exponentially space bounded ATM $\mathcal{M} = (Q, \Lambda, \Gamma, q_0, \Delta)$ whose word problem is 2-EXPTIME-hard [Chandra et al. 1981]. Our aim is to reduce the word problem of Mto node satisfiability in $\mathsf{CoreXPath}_{\downarrow,\uparrow}(\cap)$. We may assume that the length of every computation of $\mathcal{M}$ on $w \in \Lambda^k$ is bounded by 2^{2^k}, and all the configurations wqw' in such computations satisfy $|ww'| \leq 2^k$. We may also assume without loss of generality that $\mathcal{M}$ never attempts to move left on the leftmost tape cell. Let $w = a_0 \cdots a_{k-1} \in \Lambda^*$ be an input to $\mathcal{M}$. We construct a node expression $\varphi_{\mathcal{M},w}$ of $\mathsf{CoreXPath}_{\downarrow,\uparrow}(\cap)$ such that $w \in L(\mathcal{M})$ if, and only if, $\varphi_{\mathcal{M},w}$ is satisfiable.

In XML trees satisfying $\varphi_{\mathcal{M},w}$, nodes are used to represent tape cells and the leafs of certain subtrees are used to represent configurations. This is illustrated in Figure 3, where the triangles denote binary trees of depth k, and the dashed boxes enclose the nodes that describe configurations. The figure shows a configuration (left dashed box) with two successor configurations (middle and right dashed box).

In the reduction, multi-labels are comprised of the labels $Q \cup \Gamma \cup \{r, c_0, \ldots, c_{k-1}\}$, whose intuitive meaning is as follows:

—$c_0, \ldots, c_{k-1}$ describe a binary counter C used for identifying the 2^k tape cells of configurations, with the leftmost cell having counter value $C = 0$ and the rightmost $C = 2^k - 1$;

—$q \in Q$ is true at a node n in a configuration if in this configuration, the head of $\mathcal{M}$ is on the tape cell represented by n and the machine is in state q;

—$a \in \Gamma$ is true in a node n of a configuration if a is the symbol on the tape cell represented by n;

—r is used to distinguish the roots of configuration trees from the intermediate nodes as shown in Figure 3; this is important for travelling to successor configurations using a path expression.

Let $\downarrow^i$ denotes the i-fold composition $\downarrow/\cdots/\downarrow$. Then, in view of Figure 3, it is clear that

—$\alpha_{\mathsf{root}} := \downarrow^*[r]$ reaches the roots of configuration trees (from the root of the XML tree),

—$\alpha_{\mathsf{cell}} := \downarrow^*[r]/\downarrow^k$ reaches all nodes in configurations,

—$\alpha_{\mathsf{cur}} := \uparrow^k/\downarrow^k$ travels between nodes of the same configuration, and

—$\alpha_{\mathsf{nxt}} := \uparrow^{k+1}/\downarrow[\neg r]/\downarrow[r]/\downarrow^k$ travels from the nodes of a configuration to the nodes of successor configurations.

We first establish, underneath each r node, a tree in which we find every tape cell (i.e., counter value of C) at least once as a leaf. Define

$$\varphi_{\mathsf{conf}} := \bigwedge_{0\le i<k} \mathsf{every}\Big(\alpha_{\mathsf{root}}/\downarrow^i, \langle\downarrow[c_i \wedge \mathsf{every}(\downarrow^*, c_i)]\rangle \wedge \langle\downarrow[\neg c_i \wedge \mathsf{every}(\downarrow^*, \neg c_i)]\rangle\Big).$$

Since we are not allowed to use sibling axes in $\mathsf{CoreXPath}_{\downarrow,\uparrow}(\cap)$, we cannot enforce that every value of C occurs *at most* once among the leafs. Instead, we ensure that cells with identical C values are labeled in the same way. The following path expression $\alpha_{=\mathsf{cur}}$ travels between any two nodes of a configuration with the same C value, and φ_{uni} ensures unique labels:

$$\begin{aligned}
\alpha_{=i} &:= (.[c_i]/\alpha_{\mathsf{cur}}[c_i]) \cup (.[\neg c_i]/\alpha_{\mathsf{cur}}[\neg c_i]) \quad \text{for all } i < k\\
\alpha_{=\mathsf{cur}} &:= \bigcap_{0\le i<k} \alpha_{=i}\\
\varphi_{\mathsf{uni}} &:= \mathsf{every}\Big(\alpha_{\mathsf{cell}}, \bigwedge_{a\in\Gamma\cup Q} \big((a \Rightarrow \mathsf{every}(\alpha_{=\mathsf{cur}}, a)) \wedge (\neg a \Rightarrow \mathsf{every}(\alpha_{=\mathsf{cur}}, \neg a))\big)\Big).
\end{aligned}$$

It is easy to construct a node expression φ_{tape} ensures that (i) every node in a configuration is marked with exactly one symbol from Γ and never with two different states and (ii) the initial configuration (whose root is reachable from the root of the XML tree by travelling $\downarrow[r]$) is such that the head is on the left-most tape cell, the ATM is in state q_0, and the tape is labeled with the input word followed by blanks. Details are left to the reader. The following node expression φ_{head} guarantees that

each configuration has at most one cell marked with the tape head:

$$\begin{aligned}
\alpha_{\neq i} &:= (.[c_i]/\alpha_{\mathsf{cur}}[\neg c_i]) \cup (.[\neg c_i]/\alpha_{\mathsf{cur}}[c_i]) \quad \text{for all } i < k \\
\alpha_{\neq \mathsf{cur}} &:= \bigcup_{0 \leq i < k} \alpha_{\neq i} \\
\varphi_{\mathsf{head}} &:= \mathsf{every}\Big(\alpha_{\mathsf{cell}}, \bigwedge_{q,q' \in Q} (q \Rightarrow \mathsf{every}(\alpha_{\neq \mathsf{cur}}, \neg q'))\Big)
\end{aligned}$$

We now say that cells not underneath the head are labeled with the same alphabet symbol in the consecutive configuration. We use the path expression $\alpha_{=\mathsf{nxt}}$, which travels from leafs of a configuration to leafs of the successor configurations that represent the same tape cell:

$$\begin{aligned}
\alpha_{=\mathsf{nxt}} &:= \bigcap_{i<k} \big((.[c_i]/\alpha_{\mathsf{nxt}}[c_i]) \cup (.[\neg c_i]/\alpha_{\mathsf{nxt}}[\neg c_i])\big) \\
\varphi_{\mathsf{id}} &:= \mathsf{every}\Big(\alpha_{\mathsf{cell}}, \bigwedge_{a \in \Gamma} \big(a \wedge (\neg \bigvee_{q \in Q} q) \Rightarrow \mathsf{every}(\alpha_{=\mathsf{nxt}}, a)\big)\Big).
\end{aligned}$$

Next, we have to ensure that the transitions are according to the transition table. To this end, we need path expressions α_{Lcur} and α_{Rcur} that are similar to $\alpha_{=\mathsf{cur}}$, but travel to the left and right neighboring cell in the current configuration. We only give α_{Rcur} explicitly:

$$\begin{aligned}
\alpha_{\mathsf{Rcur}} &:= \bigcap_{i<k} (\alpha_{\mathsf{flip}\text{-}i} \cup \alpha_{\mathsf{keep}-i}) \\
\alpha_{\mathsf{flip}-i} &:= .[c_0 \wedge \cdots \wedge c_{i-1}]/\alpha_{\neq i} \\
\alpha_{\mathsf{keep}-i} &:= .[\neg c_0 \vee \cdots \vee \neg c_{i-1}]/\alpha_{=i}.
\end{aligned}$$

For $\alpha_{\mathsf{flip}-i}$ and $\alpha_{\mathsf{keep}-i}$, note that the conjunction of zero formulas is a tautology, and the disjunction of zero formulas a contradiction. Now, the following node expression takes care of proper transitions:

$$\begin{aligned}
\varphi_\Delta := \mathsf{every}\Big(\alpha_{\mathsf{cell}}, &\bigwedge_{q \in Q_\exists, a \in \Gamma} (q \wedge a \Rightarrow \bigvee_{(p,b,M) \in \Delta(q,a)} \langle \alpha_{=\mathsf{nxt}}[b \wedge \mathsf{every}(\alpha_{M\mathsf{cur}},\ p)]\rangle) \wedge \\
&\bigwedge_{q \in Q_\forall, a \in \Gamma} (q \wedge a \Rightarrow \bigwedge_{(p,b,M) \in \Delta(q,a)} \langle \alpha_{=\mathsf{nxt}}[b \wedge \mathsf{every}(\alpha_{M\mathsf{cur}},\ p)]\rangle)\Big).
\end{aligned}$$

It remains to describe acceptance of the machine. Since all computations of $\mathcal{M}$ are finite, it suffices to require that the rejecting state q_r never appears: $\varphi_{\mathsf{acc}} := \mathsf{every}(\alpha_{\mathsf{cell}}, \neg q_r)$. Altogether, the machine's behavior is described by the node expression

$$\varphi_{\mathcal{M},w} := \varphi_{\mathsf{conf}} \wedge \varphi_{\mathsf{uni}} \wedge \varphi_{\mathsf{tape}} \wedge \varphi_{\mathsf{head}} \wedge \varphi_{\mathsf{id}} \wedge \varphi_\Delta \wedge \varphi_{\mathsf{acc}}.$$

It is possible to show that $w \in L(\mathcal{M})$ if, and only if, $\varphi_{\mathcal{M},w}$ is satisfiable, which establishes the following result.

THEOREM 27. *Path containment for* $\mathsf{CoreXPath}_{\downarrow,\uparrow}(\cap)$ *is 2-*EXPTIME*-hard.*

6.3. $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$ IS 2-EXPTIME-hard. The proof is similar to the one given in the previous section. In particular, it is by reduction of the word problem of the

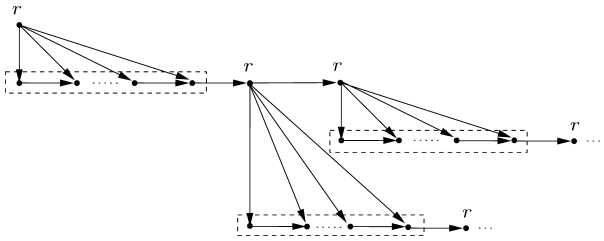


FIG. 4. Successor configurations in $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$.

same ATM $\mathcal{M}$. The main difference between this reduction and the previous one is that, in $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$, configurations have to be represented in a different way. This is illustrated in Figure 4, where each horizontal sequence enclosed by a dashed box is of length 2^k and represents a configuration. As before, the figure shows one configuration (topmost dashed box) together with two successor configurations (middle and bottommost dashed box). In the reduction, we use the same labels as in the previous section, except that an additional symbol $m_{M,q}$ is introduced for each $M \in \{L, R\}$ and $q \in Q$. The new symbols serve as markers, to be explained later on. As before, we use the counter C to identify tape cells. The r marker identifies the roots of configurations, and all elements of dashed boxes satisfy $\neg r$. In view of Figure 4, it is clear that

—$\alpha'_{\mathsf{root}} := \downarrow^*[r]$ reaches all nodes whose children represent a configuration,
—$\alpha'_{\mathsf{cell}} := \downarrow^*[\neg r]$ reaches all nodes representing tape cells, and
—$\alpha'_{\mathsf{nxt}} := \rightarrow^+[r]/\downarrow$ travels from the nodes of a configuration to the nodes of successor configurations.

Note that there is no direct counterpart for the path expression α_{cur} from the previous reduction: since we cannot travel up or left, we can only reach those cells of the same configuration that are to the right using

$$\alpha'_{>\mathsf{cur}} := \rightarrow^+,$$

but there is no way to reach those cells of the same configuration that are to the left. Fortunately, this is not essential for the reduction.

The node expression $\varphi'_{\mathcal{M},w}$ that we construct for an input w to $\mathcal{M}$ is rather similar to the node expression $\varphi_{\mathcal{M},w}$ from the previous reduction. In particular, it is a conjunction whose conjuncts correspond to those in $\varphi_{\mathcal{M},w}$. We use the same names to denote conjuncts of $\varphi_{\mathcal{M},w}$ and $\varphi'_{\mathcal{M},w}$, but add a prime for the latter.

We start with describing φ'_{conf}, which sets up the counter and the r marker, enforcing that the tape cells (labeled with $\neg r$) are always to the left of the roots of successor configurations (labeled with r). In contrast to the previous reduction, we are even able to enforce that every counter value of C occurs exactly once in every configuration:

$$\begin{aligned}\varphi'_{\mathsf{conf}} := \ & \mathsf{every}(\alpha'_{\mathsf{root}}, \langle\downarrow[\neg c_0 \wedge \cdots \wedge \neg c_{k-1} \wedge \neg r]\rangle) \wedge \\ & \mathsf{every}(\alpha'_{\mathsf{cell}}, (\neg c_0 \vee \cdots \vee \neg c_{k-1}) \Rightarrow \langle\alpha'_{\mathsf{Rcur}}[\neg r]\rangle) \wedge \\ & \mathsf{every}(\alpha'_{=\mathsf{cur}}, \bot) \\ & \mathsf{every}(\alpha'_{\mathsf{root}}/\downarrow[r]/\rightarrow^+, r)\end{aligned}$$

where α'_{Rcur} relates each node with counter value C to those nodes that are to the right and have counter value $C + 1$, and likewise for $\alpha'_{=\mathsf{cur}}$ and nodes that have identical counter values. Formally, α'_{Rcur} and $\alpha'_{=\mathsf{cur}}$ are obtained from α_{Rcur} and $\alpha_{=\mathsf{cur}}$ in the previous section by replacing $\alpha_{=i}$ with

$$\alpha'_{=i} := (.[c_i]/\alpha'_{>\mathsf{cur}}[c_i]) \cup (.[\neg c_i]/\alpha'_{>\mathsf{cur}}[\neg c_i]),$$

and likewise for $\alpha_{\neq i}$. The node expressions φ'_{uni}, φ'_{head}, φ'_{id}, and φ'_{acc} are simply the corresponding formulas from the previous section, modified by replacing

—α_{cell} with α'_{cell}, α_{nxt} with α'_{nxt}, and
—$\alpha_{=i}$ and $\alpha_{\neq i}$ as described above.

The node expression φ'_Δ needs slightly more effort. Let the path expression $\alpha'_{=\mathsf{nxt}}$ be obtained from the corresponding unprimed one by the above replacements. Then, define

$$\varphi'_\Delta := \mathsf{every}\Big(\alpha'_{\mathsf{cell}}, \bigwedge_{q\in Q_\exists, a\in\Gamma}\Big(q \wedge a \Rightarrow \bigvee_{(p,b,M)\in\Delta(q,a)} \langle\alpha'_{=\mathsf{nxt}}[b \wedge m_{M,p}]\rangle\Big) \wedge \bigwedge_{q\in Q_\forall, a\in\Gamma}\Big(q \wedge a \Rightarrow \bigwedge_{(p,b,M)\in\Delta(q,a)} \langle\alpha'_{=\mathsf{nxt}}[b \wedge m_{M,p}]\rangle\Big)\Big)$$

To make φ'_Δ work, we need to implement the intended behaviour of the markers $m_{M,q}$, namely that if $m_{M,q}$ holds in a tape cell, then the tape cell that is reached by travelling $M \in \{L, R\}$ satisfies q. This is done as follows:

$$\varphi'_{\mathsf{mark}} := \mathsf{every}\Big(\alpha'_{\mathsf{cell}}, \bigwedge_{q\in Q}\big(\big(\langle\alpha'_{\mathsf{Rcur}}[m_{L,q}]\rangle \Rightarrow q\big) \wedge \big(m_{R,q} \Rightarrow \langle\alpha'_{\mathsf{Rcur}}[q]\rangle\big)\big)\Big)$$

The two conjuncts in the formula state that if the cell to the right is labeled $m_{L,q}$, then the current cell is labeled q; and if the current cell is labeled $m_{R,q}$, then the cell to the right is labeled q. Altogether, the machine's behavior is described by the node expression

$$\varphi'_{\mathcal{M},w} := \varphi'_{\mathsf{conf}} \wedge \varphi'_{\mathsf{uni}} \wedge \varphi'_{\mathsf{tape}} \wedge \varphi'_{\mathsf{head}} \wedge \varphi'_{\mathsf{id}} \wedge \varphi'_\Delta \wedge \varphi'_{\mathsf{mark}} \wedge \varphi'_{\mathsf{acc}}$$

Again, it is possible to show that $w \in L(\mathcal{M})$ if, and only if, $\varphi'_{\mathcal{M},w}$ is satisfiable. Observe that we did not use the nontransitive sibling axis "$\rightarrow$", as promised in Section 2.2.

THEOREM 28. *Path containment for* $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$ *is 2-*EXPTIME*-hard.*

6.4. $\mathsf{CoreXPath}_{\downarrow}(\cap)$ IS EXPSPACE-HARD. Finally, we consider the *downwards fragment* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ and provide a matching EXPSPACE lower bound for Theorem 24. The proof is by a reduction of the word problem of exponentially *time*-bounded ATMs. According to Chandra et al. [1981], there is an exponentially time bounded ATM $\mathcal{M}'$ whose word problem is EXPSPACE-hard. We may assume that the length of every computation of $\mathcal{M}'$ on $w \in \Lambda^k$ is bounded by 2^k, and that all configurations wqw' in such computations satisfy $|ww'| \leq 2^k$. As in the previous reductions, we also assume that $\mathcal{M}'$ never attempts to move left on the leftmost tape cell. Let $w = a_0 \cdots a_{k-1} \in \Lambda^*$ be an input to $\mathcal{M}'$. We sketch the construction of a node expression $\varphi_{\mathcal{M}',w}$ of $\mathsf{CoreXPath}_{\downarrow}(\cap)$ such that $w \in L(\mathcal{M}')$ if, and only if, $\varphi_{\mathcal{M}',w}$ is satisfiable.

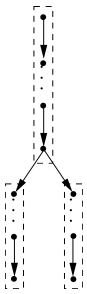


FIG. 5. Successor configurations in $\mathsf{CoreXPath}_{\downarrow}(\cap)$.

Figure 5 shows how we represent a configuration and two successor configurations. Each dashed box encloses a configuration, which is represented by a downward sequence of length 2^k. The two lower boxes represent the successor configurations of the upper box. We use the same symbols as in the previous reduction (except r), and additionally introduce labels $d_0, \ldots, d_{k-1}$ for implementing a second counter D. While the purpose of C is still to identify the tape cells of a configuration, the purpose of D is to identify configurations. In view of Figure 5, it is clear that

—$\alpha''_{\mathsf{cell}} := \downarrow^*$ reaches all nodes representing tape cells,

—$\alpha''_{>\mathsf{cur}}$, defined as follows, travels from all nodes of a configuration to the nodes of the same configuration that are below it:

$$\alpha''_{=i/D} := (.[d_i]/\downarrow^*[d_i]) \cup (.[\neg d_i]/\downarrow^*[\neg d_i])$$

$$\alpha''_{>\mathsf{cur}} := \bigcap_{i<k} \alpha''_{=i/D}$$

—α''_{nxt} travels from the nodes of a configuration to the nodes of successor configurations:

$$\alpha''_{\mathsf{nxt}} := \downarrow^* \cap \bigcap_{i<k} (\alpha''_{\mathsf{flip}\text{-}i/D} \cup \alpha''_{\mathsf{keep}-i/D})$$

where $\alpha''_{\mathsf{flip}\text{-}i/D}$ and $\alpha''_{\mathsf{keep}-i/D}$ are defined analogously to $\alpha_{\mathsf{flip}\text{-}i}$ and $\alpha_{\mathsf{keep}-i}$ from Section 6.2, but for the counter D rather than for C.

Note that the counter D is needed to define the last two path expressions. Intuitively, this is the reason why it is not possible to reduce the word problem of exponentially *space*-bounded Turing machines and show 2-EXPTIME-hardness.

Once more, the structure of the node expression $\varphi''_{\mathcal{M}',w}$ that we construct for an input w to $\mathcal{M}'$ is rather similar to the node expression $\varphi_{\mathcal{M},w}$ from the reduction in Section 6.2. In particular, it is again a conjunction whose conjuncts correspond to those in $\varphi_{\mathcal{M},w}$. We identify the conjuncts of $\varphi''_{\mathcal{M}',w}$ by a double dash. The formula $\varphi''_{\mathsf{conf}}$ sets up the counters. It says that

—both counters are initialized with 0 at the root of the tree;

—if at least one of the counters does not have maximum value, there is a successor;

—all successors have their C-values increased by one modulo 2^k;

—if the C-value is maximum, all successors have their D-values increased by one, and otherwise the D-value does not change.

We leave details to the reader. Note that we can use the nontransitive axis "$\downarrow$" in $\varphi''_{\mathsf{conf}}$ since it is included in the original version of CoreXPath, unlike the nontransitive sibling axes. The conjunct φ_{uni} is not needed since $\varphi''_{\mathsf{conf}}$ ensures that every counter value occurs exactly once in every configuration. The conjuncts φ_{tape} and φ_{acc} are readily adapted to the new encoding. For the remaining conjuncts, we proceed as follows:

—$\varphi''_{\mathsf{head}}$ is obtained from φ_{head} by replacing α_{cell} with α''_{cell} and $\alpha_{\neq \mathsf{cur}}$ with $\alpha''_{>\mathsf{cur}}$;
—φ''_{id} is obtained from φ_{id} by replacing the node expression α_{cell} with α''_{cell}, and the path expression α_{nxt} (inside $\alpha_{=\mathsf{nxt}}$) with α''_{nxt};
—φ''_{Δ} and $\varphi''_{\mathsf{mark}}$ are obtained from φ'_{Δ} and φ'_{mark} by replacing α'_{cell} with α''_{cell}, α'_{nxt} (inside $\alpha'_{=\mathsf{nxt}}$) with α''_{nxt}, and α'_{Rcur} with $\downarrow$.

Altogether, the machine's behavior is described by the node expression

$$\varphi''_{\mathcal{M}',w} := \varphi''_{\mathsf{conf}} \wedge \varphi''_{\mathsf{tape}} \wedge \varphi''_{\mathsf{head}} \wedge \varphi''_{\mathsf{id}} \wedge \varphi''_{\Delta} \wedge \varphi''_{\mathsf{mark}} \wedge \varphi''_{\mathsf{acc}}.$$

Again, it is standard to show that $w \in L(\mathcal{M}')$ if, and only if, $\varphi''_{\mathcal{M}',w}$ is satisfiable.

THEOREM 29. *Path containment for* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *is* EXPSPACE*-hard.*

7. *Path Complementation and For-Loops*

We consider CoreXPath(−) and CoreXPath(for), which extend CoreXPath with path complementation and for loops, respectively. We show that for both variants of CoreXPath, path containment is nonelementary, that is, the time needed to solve this problem cannot be bounded by any exponential tower of constant height.

We start with CoreXPath(−). Path containment is nonelementary even for a small fragment F of CoreXPath(−), where path expressions are formed as follows:

$$\alpha, \beta \ ::= \ \downarrow[p] \ \mid \ \downarrow^* \ \mid \ \alpha/\beta \ \mid \ \alpha - \beta,$$

where p ranges over $\Sigma \cup \{\top\}$. Note that F has only downward axes, lacks complex filter expressions, and lacks union as a primitive operator.

THEOREM 30. *Path containment for* F *is nonelementary.*

PROOF. We give a reduction from the non-emptiness problem for *star-free expressions* (denoted below by r and s), which are built according to the syntax rule

$$r, s := a \mid (rs) \mid (r \cup s) \mid -r,$$

where a ranges over labels from the alphabet Σ. Each star-free expression r defines a set of words $\mathcal{L}(r)$ over Σ: $\mathcal{L}(a) = \{a\}$, $\mathcal{L}(rs) = \mathcal{L}(r) \cdot \mathcal{L}(s)$, $\mathcal{L}(r \cup s) = \mathcal{L}(r) \cup \mathcal{L}(s)$, and $\mathcal{L}(-r) = \Sigma \setminus \mathcal{L}(r)$. The nonemptiness problem for this kind of expression is well known to be nonelementary [Stockmeyer 1974]. First, note that using the path complementation operator, we can define path union (via intersection):

$$\begin{aligned} \alpha \cap \beta &\equiv (\alpha - (\alpha - \beta)) \\ \alpha \cup \beta &\equiv \downarrow^* - ((\downarrow^* - \alpha) \cap (\downarrow^* - \beta)) \end{aligned}$$

Star-free expressions can now be translated into F, by the function tr defined as

follows:

$$\begin{array}{lcl} \mathsf{tr}(a) & = & \downarrow[a] \\ \mathsf{tr}(rr') & = & \mathsf{tr}(r)/\mathsf{tr}(r') \\ \mathsf{tr}(r \cup r') & = & \mathsf{tr}(r) \cup \mathsf{tr}(r') \\ \mathsf{tr}(-r) & = & \downarrow^+ - \mathsf{tr}(r). \end{array}$$

Note that $\downarrow^+$ can be defined as $\downarrow[\top]/\downarrow^*$. We can show by induction on r that, for any XML tree $T = (N, R_\downarrow, R_\rightarrow, L)$ with nodes n, m, and for any star-free expression r, $(n, m) \in [\![\mathsf{tr}(r)]\!]^T_{\mathsf{PExpr}}$ if, and only if, there are $n_1 R_\downarrow n_2 R_\downarrow \cdots R_\downarrow n_k$ such that $n_1 = n$, $n_k = m$, and the word $(L(n_2), \ldots, L(n_k))$ belongs to the language $\mathcal{L}(r)$. It follows that r defines a nonempty language if, and only if, $\mathsf{tr}(r)$ is not contained in $\downarrow^* - \downarrow^*$. The exponential blowup involved in the definition of union is of no importance since our intention is only to show nonelementarity. □

It follows that node satisfiability is also nonelementary for F. This improves on a result from Hidders [2003], which shows that F-satisfiability is NP-hard.

Next, we consider CoreXPath(for). The for-construct in XPath 2.0 allows iteration over a node set, using a bound variable. Formally, CoreXPath(for) is obtained by introducing a countably infinite set of node variables $\$i, \$j, \ldots$, and extending the syntax and semantics of CoreXPath in the following way:

—All node and path expressions are interpreted *relative to an assignment g of nodes to the variables*, that is, we replace $[\![\cdot]\!]^T_{\mathsf{NExpr}}$ and $[\![\cdot]\!]^T_{\mathsf{PExpr}}$ with $[\![\cdot]\!]^{T,g}_{\mathsf{NExpr}}$ and $[\![\cdot]\!]^{T,g}_{\mathsf{PExpr}}$, respectively. The semantics of the existing CoreXPath expressions does not change and simply ignores the additional argument.

—We allow filter expressions of the form ". is $\$i$", interpreted as follows:

$$[\![\text{. is } \$i]\!]^{T,g}_{\mathsf{NExpr}} = \{n \mid n = g(\$i)\}$$

—We allow path expressions of the form "for $\$i$ in α return β", interpreted as follows:

$$[\![\text{for } \$i \text{ in } \alpha \text{ return } \beta]\!]^{T,g}_{\mathsf{PExpr}} = \big\{(n, m) \mid \text{there is a node } k \text{ such that } (n, k) \in [\![\alpha]\!]^{T,g}_{\mathsf{PExpr}} \text{ and } (n, m) \in [\![\beta]\!]^{T,g[\$i \mapsto k]}_{\mathsf{PExpr}}\big\}$$

with $g[\$i \mapsto k]$ the assignment that agrees with g on all node variables except $\$i$, and that sends $\$i$ to k.

As usual, the downward fragment of CoreXPath(for) is denoted by $\mathsf{CoreXPath}_\downarrow(\mathsf{for})$.

THEOREM 31. *Path containment for* $\mathsf{CoreXPath}_\downarrow(\mathsf{for})$ *is nonelementary, even if only one variable is admitted.*

PROOF. Using a single variable, we can express complementation: if α, β are downward path expressions, then $\alpha - \beta$ is equivalent to

$$\text{for } \$i \text{ in } \alpha \text{ return } .[\neg\langle\beta[\text{. is } \$i]\rangle]/\downarrow^*[\text{. is } \$i].$$

Intuitively, i binds to each node reachable via α, the filter expression $[\neg\langle\beta[\text{. is } \$i]\rangle]$ filters out nodes reachable by β, and the expression $\downarrow^*[\text{. is } \$i]$ actually travels to i.

It follows that $\mathsf{CoreXPath}_\downarrow(\mathsf{for})$ is at least as complex as $\mathsf{CoreXPath}_\downarrow(-)$. □

8. *Succinctness*

We establish a number of results on the relative succinctness of different extensions of CoreXPath, in particular CoreXPath, CoreXPath($\approx$), CoreXPath($\cap$), and the extension of these three languages with transitive closure.

Suppose two languages, L and L', have the same expressive power. Upper bounds on the relative succinctness of L and L' are proved simply by exhibiting a translation that involves only a limited blowup. Establishing a lower bound means to prove the existence of expressions $(\varphi_i)_{i\in\mathbb{N}}$ such that each φ_i is of length polynomial in i and every sequence of equivalent L'-expressions grows fast in i. We say that L is *exponentially more succinct* than L' if there is a lower bound where this growth is exponential and that L is *exactly* exponentially more succinct than L' if, additionally, there is a matching upper bound. For nonelementary succinctness, we use analogous terminology.

In this section, we first concentrate on upper bounds and then on lower ones. We start with an exponential translation from CoreXPath($\cap$) to CoreXPath. This is matched by a lower bound proved later in this section, which states that CoreXPath($\cap$) is (at least) exponentially more succinct than CoreXPath($\approx$). In summary, we have thus shown that CoreXPath($\cap$) is exactly exponentially more succinct than both CoreXPath and CoreXPath($\approx$).

THEOREM 32. *There is a single exponential translation from* CoreXPath($\cap$) *path expressions to* CoreXPath *path expressions*.

PROOF SKETCH. The proof is along the same lines as in Benedikt et al. [2005], where it is shown that positive path expressions of CoreXPath($\cap$) can be translated to positive path expressions of CoreXPath with an exponential blowup. We call a path expression positive if it does not contain any negation signs.

We show how to do the translation for *basic* CoreXPath($\cap$) expressions α, that is, expressions that do not contain $\cap$ inside the φ part of sub-expressions $\beta[\varphi]$. In the general case, the translation is applied repeatedly in a bottom-up fashion.

Let α be a basic CoreXPath($\cap$) expression and let $\beta_1[\psi_1], \ldots, \beta_k[\psi_k]$ be all subexpressions of α of the form $\beta[\varphi]$. Consider the first-order language Lwith the vocabulary $R_\downarrow$, $R_{\downarrow^*}$, $R_\rightarrow$, $R_{\rightarrow^*}$, and $P_{\psi_1}, \ldots, P_{\psi_k}$. Lis interpreted in XML trees and each predicate P_{ψ_i} is interpreted in the same way as ψ_i. As shown in Benedikt et al. [2005], α can be translated in linear time into a positive existential $\mathcal{L}$-formula $\varphi(x, y)$ such that α and $\varphi(x, y)$ define the same binary relation. As shown in Benedikt et al. [2008] and Gottlob et al. [2006], $\varphi(x, y)$ can be translated in exponential time into a (positive) CoreXPath expression γ over the set of node labels $P_{\psi_1}, \ldots, P_{\psi_k}$. It remains to replace each P_{ψ_i} in γ with ψ_i. □

If we add transitive closure to the three languages mentioned above, the situation is less clear. In fact, we do not know whether it is at all possible to translate CoreXPath($*, \cap$) path expressions to CoreXPath($*$) in an equivalence-preserving way. However, there is such a translation from CoreXPath($*, \cap$) to CoreXPath($*, \approx$). This follows from the single exponential translation from CoreXPath($*, \cap$) to $\text{CoreXPath}_{\text{NFA}}(*, \text{loop}, \text{let})$ given in in Section 4, together with the following lemma, which establishes a single exponential translation from $\text{CoreXPath}_{\text{NFA}}(*, \text{loop}, \text{let})$ to $\text{CoreXPath}_{\text{NFA}}(*, \text{loop})$. Note that, in contrast to the translation in the proof of Lemma 18, the following translation is equivalence-preserving rather than only satisfiability-preserving.

LEMMA 33.

(1) *Every* $\mathsf{CoreXPath}_{\mathrm{NFA}}(*, \mathsf{loop}, \mathsf{let})$ *node expression* φ *is equivalent to a* $\mathsf{CoreXPath}(*, \approx)$ *node expression of length at most* $2^{7|\varphi|}$.

(2) *Every* $\mathsf{CoreXPath}_{\mathrm{NFA}}(*, \mathsf{loop}, \mathsf{let})$ *path automaton* π *is equivalent to a* $\mathsf{CoreXPath}(*, \approx)$ *path expression of length at most* $2^{7|\pi|}$.

(3) *Every* $\mathsf{CoreXPath}_{\mathrm{NFA}}(*, \mathsf{loop}, \mathsf{let})$ *EPA* (π, ρ) *is equivalent to a* $\mathsf{CoreXPath}(*, \approx)$ *path expression of length at most* $2^{7(|\pi|+|\rho|)}$.

PROOF. Items 1 and 2 are proved by simultaneous induction, while item 3 follows from 1 and 2 by a simple counting argument.

(1) *Node Expressions*. The induction start is trivial because atomic $\mathsf{CoreXPath}_{\mathrm{NFA}}(*, \mathsf{loop}, \mathsf{let})$ node expressions are also atomic $\mathsf{CoreXPath}(*, \approx)$ node expressions. In the induction step, the only nontrivial cases are for node expressions of the form $\mathsf{loop}(\pi)$ and $\mathsf{let}\ p := \psi\ \mathsf{in}\ \chi$. First suppose that φ is of the form $\mathsf{loop}(\pi)$. By the induction hypothesis, π is equivalent to a $\mathsf{CoreXPath}(*, \approx)$ path expression α of length $2^{7|\pi|}$. It follows that $\mathsf{loop}(\pi)$ is equivalent to $\alpha \approx .$, which has length at most $2^{7|\pi|} + 2$, which is less than $2^{7|\varphi|}$.

Now suppose that φ is of the form $\mathsf{let}\ p := \psi\ \mathsf{in}\ \chi$. Recall that, by the definition of formula size, $|\varphi| = |\psi| + |\chi| + 1$. By the induction hypothesis, ψ and χ are equivalent to $\mathsf{CoreXPath}(*, \approx)$ node expressions ψ' and χ' of length at most $2^{7|\psi|}$ and $2^{7|\chi|}$, respectively. Let φ' be the result of replacing in ψ' every occurrence of p with χ'. Then, φ' is equivalent to φ, and $|\varphi'| \leq |\psi'| \cdot |\chi'| \leq 2^{7|\psi'|} \cdot 2^{7|\chi|} \leq 2^{7|\varphi|}$.

(2) *Path Automata*. Consider any $\mathsf{CoreXPath}_{\mathrm{NFA}}(*, \mathsf{loop}, \mathsf{let})$ path automaton $\pi = (Q, \Delta, q_I, q_F)$ with $|Q| = m$. Recall that $|\pi| = m + \sum_{(q,.[\varphi],q') \in \Delta} |\varphi|$. For each transition $(q, .[\varphi], q') \in \Delta$, we use the induction hypothesis to obtain a $\mathsf{CoreXPath}(*, \approx)$ node expression φ' equivalent to φ and of length at most $2^{7|\varphi|}$. We then use a standard construction to convert π into an equivalent regular expression of size at most 2^{7m} (in fact, 2^{4m+3}) cf. McNaughton and Yamada [1960], and Ellul et al. [2004]. We obtain a $\mathsf{CoreXPath}(*, \approx)$ path expression of length $2^{7m} \cdot 2^{7 \max\{|\varphi||(q,.[\varphi],q') \in \Delta\}}$, which is bounded by $2^{7|\pi|}$.

(3) *Extended Path Automata*. Consider any $\mathsf{CoreXPath}_{\mathrm{NFA}}(*, \mathsf{loop})$ EPA (π, ρ) with $\rho = ((p_1, \varphi_1), \ldots, (p_n, \varphi_n))$. Recall that $|(\pi, \rho)| = |\pi| + \sum_{1 \leq i \leq n} (\varphi_n + 1)$. By (1) and (2), we can find a $\mathsf{CoreXPath}(*, \approx)$ path expression α of length at most $2^{7|\pi|}$ equivalent to π, and we can find, for $1 \leq i \leq n$, a $\mathsf{CoreXPath}(*, \approx)$ node expression φ_i' of length at most $2^{7|\varphi_i|}$ that is equivalent to φ_i. Let α' be obtained from α by replacing, for $1 \leq i \leq n$, p_i with φ_i' (in the appropriate order, that is, starting with p_n). Then, $|\alpha'| \leq |\alpha| \cdot 2^{7|\varphi_1|} \cdot \ldots \cdot 2^{7|\varphi_n|} \leq |\alpha| \cdot 2^{7|\rho|}$. □

By combining Lemma 16, Lemma 17, and Lemma 33, we obtain the following result.

THEOREM 34. *There is a double exponential translation from* $\mathsf{CoreXPath}\ (*, \cap)$ *to* $\mathsf{CoreXPath}\ (*, \approx)$. *The translation is single exponential if there is a fixed bound on the intersection depth of the input expression.*

In the conference version of the present paper, we claimed that $\mathsf{CoreXPath}(*, \cap)$ is exactly exponentially more succinct than $\mathsf{CoreXPath}(*, \approx)$. However, there was a flaw in the argument for the upper bound. We currently do not know whether there is a single exponential translation from $\mathsf{CoreXPath}(*, \cap)$ to $\mathsf{CoreXPath}(*, \approx)$,

although the recent results in Gelade and Neven [2008] may suggest a negative answer. We do know that there can be no polynomial translation, by Theorem 35 below.

We now turn to lower bounds on succinctness. We start with proving the two lower bounds that have already been mentioned above.

THEOREM 35.

—CoreXPath($\cap$) *is exponentially more succinct than* CoreXPath($\approx$).

—CoreXPath($*, \cap$) *is (at least) exponentially more succinct than* CoreXPath ($*, \approx$).

PROOF. We may, without loss of generality, restrict our attention to any subset of the set of all finite XML trees. Let $\mathcal{T}^1_{p,q}$ be the class of XML trees where each node has at most one child, and each node is labeled by either p or q. In other words, $\mathcal{T}^1_{p,q}$ is the set of all words over the alphabet $\{p, q\}$. Given such a word, we use n to denote the number of nodes in the tree and u_i to denote the ith node counting from the root (for $1 \leq i \leq n$). For each $k \in \mathbb{N}$, let φ_k be the following property, where $\equiv$ means that two nodes have the same label:

> For all $i, j \leq n - 2k$, if $u_i, u_{i+1}, u_j, u_{j+1}$ all have label p, and $u_{i+2\ell} \equiv u_{j+2\ell}$ for all $\ell < k$, then $u_{i+2k} \equiv u_{j+2k}$.

CLAIM 1. *On $\mathcal{T}^1_{p,q}$, φ_k can be expressed by a node expression of* CoreXPath($\cap$) *of size quadratic in k.*

To see this, first note that $\equiv$ is defined by the path expression $(.[p]/\uparrow^*/\downarrow^*[p]) \cup (.[q]/\uparrow^*/\downarrow^*[q])$ and $\not\equiv$ is defined by the path expression $(.[p]/\uparrow^*/\downarrow^*[q) \cup (.[q]/\uparrow^*/\downarrow^*[p])$. Next, let α_ℓ and $\alpha_\ell^\smile$ be the path expressions $(\downarrow)^{2\ell}/ \equiv /(\uparrow)^{2\ell}$ and $(\downarrow)^{2\ell}/ \not\equiv /(\uparrow)^{2\ell}$, defining the binary relation that holds between u_i and u_j if $i, j \leq n - 2\ell$ and $u_{i+2\ell} \equiv u_{j+2\ell}$ (respectively, $u_{i+2\ell} \not\equiv u_{j+2\ell}$). Finally, the node expression φ_k is

$$p \wedge \langle\downarrow[p]\rangle \rightarrow \neg\left\langle(\bigcap_{\ell<k} \alpha_\ell \cap \alpha_k^\smile)[p \wedge \langle\downarrow[p]\rangle]\right\rangle.$$

CLAIM 2. *Every* CoreXPath($*, \approx$) *node expression, and in fact every* CoreXPath$_{\mathsf{NFA}}$($*$, loop) *node expression expressing φ_k on $\mathcal{T}^1_{p,q}$ must be of length $O(2^k)$.*

We have already seen that every CoreXPath$_{\mathsf{NFA}}$($*$, loop) node expression can be translated in polynomial time into a 2ATA. Since we are working with words, we can translate into a two-way alternating Büchi automaton on words rather than on trees. Each such automaton can be translated into an equivalent NFA at the cost of a single exponential blowup [Vardi 1998]. Now, Etessami et al. [2002] proved that any automaton of the latter kind defining φ_k has at least 2^{2^k} many states. The claim follows. □

In the remainder of this section, we also consider extensions of CoreXPath with path complementation and for-loops. We start with showing that the jump in complexity encountered when adding path complementation is also reflected in the succinctness.

THEOREM 36. CoreXPath($*$, $-$) *is nonelementarily more succinct than* CoreXPath($*$, $\cap$)

PROOF. It follows from Theorem 30 that the size of the smallest XML tree for a satisfiable CoreXPath($-$) node expression φ cannot be bounded elementarily in the length of φ: if there was an elementary bound, then this would easily yield an elementary decision procedure for testing CoreXPath($-$) satisfiability, that is, enumerate and try all XML trees. This means that there is a sequence of CoreXPath($-$) formulas $(\varphi_i)_{i\in\mathbb{N}}$ of length linear in i and such that each φ is satisfiable, but the smallest XML tree satisfying φ_i cannot be bounded by an elementary function of i. On the other hand, every satisfiable CoreXPath($*$, $\cap$) expression φ is satisfied in an XML tree of size at most quadruple exponential in the length of φ. This can be seen as follows. By Theorem 34, every CoreXPath($*$, $\cap$) expression can be translated into a CoreXPath($*$, $\approx$) expression with a double exponential blowup. As shown in Section 3, every expression of the latter kind can be converted into a 2ATA with only linear blowup. It is standard to show that every 2ATA A with $L(A) \neq \emptyset$ accepts an XML tree T of size double exponential in the size of A. Indeed, this easily follows from the proof of Theorem 13 and corresponding results for standard 2ATAs, see Grädel et al. [2002]. In summary, it follows that any sequence $(\varphi'_i)_{i\in\mathbb{N}}$ of CoreXPath($*$, $\cap$) formulas such that each φ'_i is equivalent to φ_i must grow nonelementarily in length. □

Our final lower bound shows that, although path complementation leads to a massive increase in succinctness, extensions with the for construct are even more succinct.

THEOREM 37. CoreXPath(for) *is (at least) exponentially more succinct than* CoreXPath($-$).

PROOF. Let FO be the first-order language, interpreted on XML trees, that has atomic binary relations $R_\downarrow$, $R_{\downarrow *}$, $R_\rightarrow$, $R_{\rightarrow^*}$ and a unary predicate P_p for each $p \in \Sigma$. There is a linear translation from CoreXPath($-$) into the three variable fragment of FO, and there is a linear translation from full FO into CoreXPath(for). It was shown in Grohe and Schweikardt [2005] that, on XML trees, FO is exponentially more succinct than its three variable fragment (in fact, the proof does not even involve unary predicates). It follows that CoreXPath(for) is also exponentially more succinct than CoreXPath($-$). □

9. *Discussion*

We have given a comprehensive complexity analysis of containment in CoreXPath extended with operators that are part of or inspired by XPath 2.0, namely path intersection, path equality, path complementation, for-loops, and transitive closure. The main outcome of our investigations is that these extensions often result in a significant increase in complexity, with the notable exception of path equality and transitive closure.

One aspect of XPath 2.0 that we have not considered is the fact that it has a *sequence-based* semantics. Unlike in XPath 1.0, where path expressions are evaluated at a context node and yield a set of nodes as a result, path expressions of XPath 2.0 yield *sequences* of nodes, possibly unordered and with duplicates. This has an

impact on static analysis as two expressions that are equivalent in the set-based semantics may no longer be equivalent under the sequence-based semantics; see also ten Cate and Marx [2009]. It is not clear to what extent the choice of semantics affects our complexity results.

Our upper bound proofs do not provide practical algorithms for query optimization in practice. Query optimization is usually implemented using rewrite rules. It is therefore natural to ask whether there is a set of rewrite rules by means of which expressions from the various dialects that we consider can be rewritten into equivalent, but more efficient expressions. In this respect, it is worth mentioning that in ten Cate and Marx [2009], a complete axiomatization for $\mathsf{CoreXPath}(\cap, -)$ and $\mathsf{CoreXPath}(\cap, -, \mathsf{for})$ is given. The axiomatizations, consisting of axioms such as $(\alpha \cup \beta)/\gamma \equiv \alpha/\gamma \cup \beta/\gamma$, are shown to be *complete* in the sense that, whenever two path expressions are equivalent, their equivalence can be proved by repeated application of the axioms.

ACKNOWLEDGMENTS. We would like to thank the anonymous reviewers of the Journal of the ACM for their very helpful comments. We also thank Stefan Göller and Maarten Marx for discussions and comments.

REFERENCES

ARENAS, M., FAN, W., AND LIBKIN, L. 2008. On verifying consistency of XML specifications. In *SIAM J. Comput. 38*, 3, 259–270.

BENEDIKT, M., FAN, W. AND GEERTS, F. 2008. XPath satisfiability in the presence of DTDs. *J. ACM 55*, 2, 1–79.

BENEDIKT, M., FAN, W., AND KUPER, G. M. 2005. Structural properties of XPath fragments. *Theoret. Comput. Sci. 336*, 1, 3–31.

BERGLUND, A., BOAG, S., CHAMBERLIN, D., FERNANDEZ, M. F., KAY, M., ROBIE, J., AND SIMEON, J., EDS. 2007. *XML Path Language (XPath) Version 2.0*. W3C Recommendation. http://www.w3.org/TR/2007/REC-xpath20-20070123/.

BOAG, S., CHAMBERLIN, D., FERNANDEZ, M. F., FLORESCU, D., ROBIE, J., AND SIMEON, J., EDS. 2007. *XQuery 1.0: an XML Query Language*. W3C Recommendation. http://www.w3.org/TR/2007/REC-xquery-20070123/.

BÖTTCHER, S. 2004. Testing intersection of XPath expressions under DTDs. In *Proceedings of the 8th International Database Engineering and Applications Symposium (IDEAS'04)*. IEEE Computer Society Press, Los Alamitos, CA, 401–406.

CHANDRA, A. K., KOZEN, D. C., AND STOCKMEYER, L. J. 1981. Alternation. *J. ACM 28*, 1, 114–133.

CLARK, J., AND DEROSE, S. J., EDS. 1999. *XML Path Language (XPath) Version 1.0*. W3C Recommendation. http://www.w3.org/TR/xpath.

CLARK, J., AND MURATA, M. 2001. RELAX NG specification. http://relaxng.org/spec-20011203.html.

DEUTSCH, A. AND TANNEN, V. 2005. XML queries and constraints, containment and reformulation. *Theoret. Comput. Sci. 336*, 1, 57–87.

ELLUL, K., KRAWETZ, B., SHALLIT, J., AND WEI WANG, M. 2004. Regular expressions: New results and open problems. *J. Automata, Lang. Combin. 9*, 2-3, 233–256.

ETESSAMI, K., VARDI, M. Y., AND WILKE, T. 2002. First order logic with two variables and unary temporal logic. *Info. Computat. 179*, 2, 279–295.

FAN, W., GEERTS, F., JIA, X., AND KEMENTSIETSIDIS, A. 2007. Rewriting regular XPath queries on XML views. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*. IEEE Computer Society Press, Los Alamitos, CA, 666–675.

FÜRER, M. 1980. The complexity of the inequivalence problem for regular expressions with intersection. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming (ICALP'80)*, J. W. de Bakker and J. van Leeuwen, Eds. Lecture Notes in Computer Science, vol. 85. Springer-Verlag, Berlin, Germany, 234–245.

GAO, S., SPERBERG-MCQUEEN, C. M., AND THOMPSON, H. S., EDS. 2007. *XML Schema Definition Language (XSDL) 1.1, Part 1: Structures*. W3C Working Draft. http://www.w3.org/TR/2007/WD-xmlschema11-1-20070830/.

GEERTS, F., AND FAN, W. 2005. Satisfiability of XPath queries with sibling axes. In *Proceedings of the 10th International Symposium on Database Programming Languages (DBPL'05)*, G. M. Bierman and C. Koch, Eds. Lecture Notes in Computer Science, vol. 3774. Springer-Verlag, Berlin, Germany, 122–137.

GELADE, W., AND NEVEN, F. 2008. Succinctness of the complement and intersection of regular expressions. In *Proceedings of the 25th International Symposium on Theoretical Aspects of Computer Science (STACS'08)*, S. Albers and P. Weil, Eds. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 325–336.

GOTTLOB, G., AND KOCH, C. 2002. Monadic queries over tree-structured data. In *Proceedings of the 17th IEEE Symposium on Logic in Computer Science (LICS'02)*, G. Plotkin, Ed. IEEE Computer Society, 189–202.

GOTTLOB, G., KOCH, C., PICHLER, R., AND SEGOUFIN, L. 2005. The complexity of XPath query evaluation and XML typing. *J. ACM 52*, 2, 284–335.

GOTTLOB, G., KOCH, C., AND SCHULZ, K. U. 2006. Conjunctive queries over trees. *J. ACM 53*, 2, 238–272.

GRÄDEL, E., THOMAS, W., AND WILKE, T., EDS. 2002. *Automata, Logics and Infinite Games*. Lecture Notes in Computer Science, vol. 2500. Springer-Verlag, Berlin, Germany.

GROHE, M., AND SCHWEIKARDT, N. 2005. The succinctness of first-order logic on linear orders. *Logical Methods in Computer Science 1*, 1, 1–25.

HAMMERSCHMIDT, B. C., KEMPA, M., AND LINNEMANN, V. 2005. On the intersection of XPath expressions. In *Proceedings of the 9th International Database Engineering and Applications Symposium (IDEAS'05)*. IEEE Computer Society Press, Los Alamitos, CA, 49–57.

HIDDERS, J. 2003. Satisfiability of XPath expressions. In *Proceedings of the 9th International Workshop on Data Base Programming Languages (DBPL'03)*, G. Lausen and D. Suciu, Eds. Lecture Notes in Computer Science, vol. 2921. Springer-Verlag, Berlin, Germany, 21–36.

KAY, M., ED. 2007. *XSL Transformations (XSLT) Version 2.0*. W3C Recommendation. http://www.w3.org/TR/2007/REC-xslt20-20070123/.

KLARLUND, N., MØLLER, A., AND SCHWARTZBACH, M. I. 2002. The DSD schema language. *Automat. Softw. Eng. 9*, 3, 285–319.

LADNER, R. E. 1977. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput. 6*, 3, 467–480.

LAKSHMANAN, L. V. S., RAMESH, G., WANG, H., AND ZHAO, Z. J. 2004. On testing satisfiability of tree pattern queries. In *Proceedings of the 30th International Conference on Very Large Databases (VLDB'04)*, M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, Eds. Morgan-Kaufmann, San Francisco, CA, 120–131.

LANGE, M., AND LUTZ, C. 2005. 2-ExpTime lower bounds for propositional dynamic logics with intersection. *J. Symb. Logic 70*, 5, 1072–1086.

MARTENS, W., NEVEN, F., SCHWENTICK, T., AND BEX, G. 2006. Expressiveness and complexity of XML schema. *ACM Trans. Datab. Syst. 31*, 3, 770–813.

MARX, M. 2004. XPath with conditional axis relations. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT'04)*, E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, and E. Ferrari, Eds. Lecture Notes in Computer Science, vol. 2992. Springer-Verlag, Berlin, Germany.

MARX, M. 2005. Conditional XPath. *ACM Trans. Datab. Syst. 30*, 4, 929–959.

MARX, M., AND DE RIJKE, M. 2005. Semantic characterizations of navigational XPath. *ACM SIGMOD Rec. 34*, 2, 41–46.

MCNAUGHTON, R., AND YAMADA, H. 1960. Regular expressions and state graphs for automata. *IEEE Trans. Electron. Comput. 9*, 39–47.

MIKLAU, G., AND SUCIU, D. 2004. Containment and equivalence for a fragment of XPath. *J. ACM 51*, 1, 2–45.

MURATA, M., LEE, D., MANI, M., AND KAWAGUCHI, K. 2005. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Tech. 5*, 4, 660–704.

NEVEN, F., AND SCHWENTICK, T. 2006. On the complexity of XPath containment in the presence of disjunction, DTDs and variables. *Logical Meth. Comput. Sci. 2*, 1–30.

OLTEANU, D., MEUSS, H., FURCHE, T., AND BRY, F. 2002. XPath: Looking forward. In *Proceedings of*

the Workshop on XML-Based Data Management (XMLDM'02), A. B. Chaudhri, R. Unland, C. Djeraba, and W. Lindner, Eds. Lecture Notes in Computer Science, vol. 2490. Springer-Verlag, Berlin, Germany, 109–127.

PAPAKONSTANTINOU, Y., AND VIANU, V. 2000. DTD inference for views of XML data. In *Proceedings of the 19th Symposium on Principles of Database Systems (PODS'00)*. ACM, New York, 35–46.

SCHWENTICK, T. 2004. XPath query containment. *SIGMOD Rec. 33*, 1, 101–109.

STOCKMEYER, L. J. 1974. The complexity of decision problems in automata theory. Ph.D. dissertation. Department of Electrical Engineering, MIT, Cambridge, MA.

TAJIMA, K., AND FUKUI, Y. 2004. Answering XPath queries over networks by sending minimal views. In *Proceedings of the 30th International Conference on Very Large Databases (VLDB'04)*, M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, Eds. Morgan Kaufmann, San Francisco, CA, 48–59.

TEN CATE, B. 2006. The expressivity of XPath with transitive closure. In *Proceedings of the 25th Symposium on Principles of Database Systems (PODS'06)*, S. Vansummeren, Ed. ACM, New York, 328–337.

TEN CATE, B., AND MARX, M. 2009. Axiomatizing the logical core of XPath 2.0. In *Proceedings of ICDT 2007*. LNCS, vol. 4353. Springer, 134–148.

THOMPSON, H. S., BEECH, D., MALLONEY, M., AND MENDELSOHN, N., EDS. 2004. *XML Schema Part 1: Structures, Second Edition*. W3C Recommendation. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

VARDI, M. 1998. Reasoning about the past with two-way automata. In *Proceedings of the 25th International Colloquium on Automata, Languages, and Programming (ICALP'98)*. Lecture Notes in Computer Science, vol. 1443. Springer-Verlag, Berlin, Germany, 628–641.

WOOD, P. T. 2003. Containment for XPath fragments under DTD constraints. In *Proceedings of the 9th International Conference on Database Technology (ICDT'03)*. Lecture Notes in Computer Science, vol. 2572. Springer-Verlag, Berlin, Germany, 300–314.

RECEIVED JANUARY 2008; REVISED MAY 2009; ACCEPTED MAY 2009