The Complexity of Query Containment in Expressive Fragments of XPath 2.0

(full version, including appendices, of the PODS'07 paper)

Balder ten Cate
ISLA – Informatics Institute
Universiteit van Amsterdam
balder.tencate@uva.nl

Carsten Lutz
Institute for Theoretical Computer Science
Dresden University of Technology
lutz@tcs.inf.tu-dresden.de

ABSTRACT

Query containment has been studied extensively for fragments of XPath 1.0. For instance, the problem is known to be EXPTIME-complete for CoreXPath, the navigational core of XPath 1.0. Much less is known about query containment in (fragments of) the richer language XPath 2.0. In this paper, we consider extensions of CoreXPath with the following operators, which are all part of XPath 2.0 (except the last): 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

General Terms
Languages, Algorithms

Keywords
XML, XPath, Containment, Satisfiability, Complexity

1. INTRODUCTION

The growing popularity of XML as a standard for representing semi-structured data has led to the definition 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. Because of the central role of XPath, the static analysis of XPath expressions is a prominent subject in research about XML processing. In particular, *containment* and *satisfiability* have been investigated for a variety of XPath fragments, and a wealth of complexity results has been obtained over the last few years (e.g., [21, 12, 1]). Many of these results also take into account a schema language, most notably DTDs, and analyze their impact on the complexity of query containment and satisfiability.

Almost all of the existing complexity results for XPath concern the 1.0 version instead of the more recent and richer XPath 2.0. The work of Hidders [12] is a notable exception, but it addresses only positive fragments of XPath 2.0, i.e., fragments in which negation of node expressions is not admitted. In this paper, we consider a family of fragments of XPath 2.0 that form a hierarchy regarding expressive power, all of them including negation. We provide a detailed analysis of the complexity of containment and related static analysis problems in these fragments, both with and without DTDs.

More specifically, we extend CoreXPath, the navigational core of XPath 1.0 [7, 8], with the following ingredients that were introduced in XPath 2.0: *path intersection* ($\cap$), *path complementation* ($-$), and *iteration* (for). Besides these three operators, we also consider *transitive closure* ($*$) and *path equalities* ($\approx$, also known as *node set equalities* and not to be confused with *data value equalities*). Path equalities are not part of XPath 2.0 as a primitive construct, but can be expressed. They have been studied in [4, 2, 22, 25]. 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, see e.g. [16, 25, 6]. 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. The expressivity hierarchy for these languages is depicted in Figure 1, based on expressivity results from [17, 18, 25].

For each of the languages shown in Figure 1, we determine the complexity of the containment problem, which ranges from EXPTIME to non-elementary. Our main results are summarized in Table 1 and Figure 1. They apply to satisfiability as well, since containment and (non-)satisfiability are polynomially inter-reducible in the XPath dialects considered here. Moreover, since DTDs can be expressed in CoreXPath($*$) with only a linear blowup in size, all upper bounds from Table 1 generalize to containment and satisfiability *in the presence of DTDs*.

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 effects

PODS'07, June 11–14, 2007, Beijing, China.

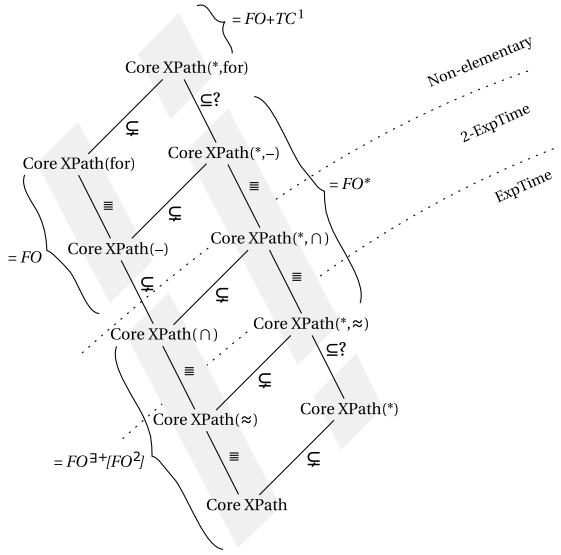


Figure 1: Hierarchy of XPath languages

of adding path complementation or for are rather devastating, as it renders containment non-elementary hard. In other words, the amenities of XPath 2.0 come at the price of a considerable 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 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 [21].

We also present some observations concerning *succinctness*. We show that CoreXPath$(\cap)$ is exactly exponentially more succinct than CoreXPath and CoreXPath$(\approx)$, explaining the higher computational complexity of the former. Likewise, CoreXPath$(*, \cap)$ is exactly exponentially more succinct than CoreXPath$(*, \approx)$. Finally, CoreXPath$(*, -)$ is non-elementarily more succinct than CoreXPath$(*, \cap)$ and CoreXPath(for) is at least exponentially more succinct than CoreXPath$(-)$.

Related work. A considerable number of papers is concerned with the complexity of static analysis for fragments of *XPath 1.0*, see [23, 4, 19, 28, 16, 21, 1]. The work of Hidders [12], which we already mentioned, studies satisfiability for positive fragments of XPath 2.0. The present paper can be seen as a continuation of the work of Hidders. Our setting is more general in several respects: (i) we consider languages that include negation of node expressions, (ii) our results also cover containment and other static analysis tasks, and (iii) we also study these tasks in the presence of a DTD. Our results also relate to [2], where closure under intersection and complementation is studied for XPath fragments that do not explicitly contain these operators.

The most important feature of XPath that we do *not* study is data value comparison. Note that already for the extension of CoreXPath with data value comparisons of the form $\alpha/@a = \beta/@b$ and $\alpha/@a =$ 'c', containment is undecidable [1].

In [26], complete axiomatizations are presented for CoreXPath$(\cap, -)$ and CoreXPath$(\cap, -,$ for$)$.

Table 1: Summary of our complexity results

The complexity of containment and satisfiability for XPath expressions:

	CoreXPath$(\ldots)$	CoreXPath$(*, \ldots)$
$\approx$	EXPTIME-complete (even for the downward fragment)	EXPTIME-complete (even for the downward fragment)
$\cap$	2-EXPTIME-complete (even for the vertical and forward fragments); EXPSPACE-complete for the downward fragment;	2-EXPTIME-complete (even for the downward fragment);
	EXPTIME-complete when the nesting depth of intersection is bounded	EXPTIME-complete when the nesting depth of intersection is bounded
$-$	Non-elementary (even for the downward fragment)	Non-elementary (even for the downward fragment)
for	Non-elementary (even for the one-variable downward fragment)	Non-elementary (even for the one-variable downward fragment)

All results also apply in the presence of DTDs.

2. PRELIMINARIES

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

2.1 Syntax and semantics of CoreXPath

An XML document, for present purposes, is a finite node-labelled sibling-ordered unranked tree.

DEFINITION 1 (TREE MODELS). *Fix a countably infinite set Σ of* labels *(or* tags*). A* tree model *is a structure $(N, R_\downarrow, R_\rightarrow, Lab)$, where $(N, R_\downarrow)$ is a finite tree (with $R_\downarrow$ the child-relation), $R_\rightarrow$ linearly orders siblings in this tree, and $Lab : N \to \Sigma$ assigns a label to each node.*

We will use $R_\uparrow$ and $R_\leftarrow$ to denote the converse of $R_\downarrow$ and $R_\rightarrow$. We do not associate data values with nodes since data value comparison is not considered in this paper.

DEFINITION 2 (SYNTAX AND SEMANTICS OF CoreXPath). *The primary expressions of* CoreXPath *are* path expressions*, which define binary relations. Inside the path expressions, one can use* node expressions*, which define sets of nodes. The two types of expressions are defined by simultaneous induction.*

- Path expressions*:* $\alpha ::= \tau \mid \tau^* \mid . \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\varphi]$
- Node expressions*:* $\varphi ::= p \mid \langle\alpha\rangle \mid \top \mid \neg\varphi \mid \varphi \wedge \psi$

where $\tau \in \{\downarrow, \uparrow, \rightarrow, \leftarrow\}$, $p \in \Sigma$, α and β are path expressions and φ and ψ are node expressions.

The semantics of CoreXPath*, relative to a tree model $M = (N, R_\downarrow, R_\rightarrow, Lab)$, is given by two functions, $[\![\cdot]\!]^M_{\mathsf{PExpr}}$ and $[\![\cdot]\!]^M_{\mathsf{NExpr}}$, mapping path expressions and node expressions to binary relations and sets, respectively. These functions are defined as follows (we omit the superscript M for readability):*

$$
\begin{array}{lcl}
[\![\tau]\!]_{\mathsf{PExpr}} & = & R_\tau \\
[\![\tau^*]\!]_{\mathsf{PExpr}} & = & \textit{the reflexive transitive closure of } [\![\tau]\!]_{\mathsf{PExpr}} \\
[\![.]\!]_{\mathsf{PExpr}} & = & \{(n,n) \mid n \in N\} \\
[\![\alpha/\beta]\!]_{\mathsf{PExpr}} & = & [\![\alpha]\!]_{\mathsf{PExpr}} \textit{ composed with } [\![\beta]\!]_{\mathsf{PExpr}} \\
[\![\alpha \cup \beta]\!]_{\mathsf{PExpr}} & = & [\![\alpha]\!]_{\mathsf{PExpr}} \cup [\![\beta]\!]_{\mathsf{PExpr}} \\
[\![\alpha[\varphi]\,]\!]_{\mathsf{PExpr}} & = & \{(n,m) \in [\![\alpha]\!]_{\mathsf{PExpr}} \mid m \in [\![\varphi]\!]_{\mathsf{NExpr}}\}
\end{array}
$$

$$
\begin{array}{lcl}
[\![p]\!]_{\mathsf{NExpr}} & = & \{n \in N \mid Lab(n) = p\} \\
[\![\langle\alpha\rangle]\!]_{\mathsf{NExpr}} & = & \{n \in N \mid \exists m \in N.\ (n,m) \in [\![\alpha]\!]_{\mathsf{PExpr}}\} \\
[\![\top]\!]_{\mathsf{NExpr}} & = & N \\
[\![\neg\varphi]\!]_{\mathsf{NExpr}} & = & N \setminus [\![\varphi]\!]_{\mathsf{NExpr}} \\
[\![\varphi \wedge \psi]\!]_{\mathsf{NExpr}} & = & [\![\varphi]\!]_{\mathsf{NExpr}} \cap [\![\psi]\!]_{\mathsf{NExpr}}
\end{array}
$$

For $\tau \in \{\downarrow, \uparrow, \rightarrow, \leftarrow\}$, we will use τ^+ as shorthand for τ/τ^* (the proper transitive closure of τ). Also, we will use $\downarrow_1$ as a shorthand for $\downarrow[\neg\langle\leftarrow\rangle]$ (the *first-child* relation), and $\uparrow_1$ for $.[\neg\langle\leftarrow\rangle]/\uparrow$ (the converse of $\downarrow_1$). Finally, we will use $\varphi \Rightarrow \psi$ as a shorthand for the node expression $\neg(\varphi \wedge \neg\psi)$.

The original version of CoreXPath as introduced in [7, 8] does not include the *non-transitive sibling axes* $\leftarrow$ and $\rightarrow$. We have included them, but all our results hold independently of whether these axes are present or not.

2.2 Extending CoreXPath

We consider extensions of CoreXPath with the following:

► *Path equalities ($\approx$).* A path equality is a node expression of the form $\alpha \approx \beta$, for α, β path expressions, 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 equalities have been studied in [4, 2, 22, 25].

► *Path intersection ($\cap$).* For any path expressions α, β,

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

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

► *Path complementation ($-$).* For any path expressions α, β,

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

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

► *Iteration (*for*).* This extension involves a countably infinite set of *node variables* $\$i, \$j, \ldots$, that can be bound using the for-construct. One can test equality of the current node and a variable $\$i$ using a node test of the form ". is $\$i$". More information on the precise syntax and semantics will be given in Section 7. For now, we only remark that path complementation can be expressed using iteration: $\alpha - \beta$ is equivalent to for $\$i$ in α return $.[\neg\langle\beta[.\text{ is }\$i]\rangle]/\uparrow^*/\downarrow^*[.\text{ is }\$i]$.

► *Transitive closure ($*$).* CoreXPath only supports the transitive closure of the atomic path expressions $\uparrow, \downarrow, \leftarrow, \rightarrow$. If transitive closure is added *as an operator on arbitrary path expressions*, paths like $(\downarrow/\downarrow)^*$ ("descendant at even distance") become expressible.

For any $X \subset \{\approx, \cap, -, \mathsf{for}, *\}$, we denote by $\mathsf{CoreXPath}(X)$ the extension of CoreXPath with the operators in X. We denote by $\mathsf{CoreXPath}_{\downarrow}(X)$ the *downward* fragment of $\mathsf{CoreXPath}(X)$, i.e., having only the downward axes $\downarrow, \downarrow^*$ (and '.'). Similarly, we denote by $\mathsf{CoreXPath}_{\downarrow\uparrow}(X)$ and $\mathsf{CoreXPath}_{\downarrow\rightarrow}(X)$ the *vertical* and *forward* fragment of $\mathsf{CoreXPath}(X)$.

2.3 Static analysis

We consider the following decision problems, which play an important role in the static analysis of XPath expressions:

► *Path containment*: given two path expressions α, β, is it true that for all tree models M, $[\![\alpha]\!]^M_{\mathsf{PExpr}} \subseteq [\![\beta]\!]^M_{\mathsf{PExpr}}$?

► *Path satisfiability*: given a path expression α, is there a tree model M such that $[\![\alpha]\!]^M_{\mathsf{PExpr}} \neq \emptyset$?

► *Node satisfiability*: given a node expression φ, is there a tree model M such that $[\![\varphi]\!]^M_{\mathsf{NExpr}} \neq \emptyset$?

For the languages we consider, these problems all have the same complexity. Indeed, the proof of the following is not difficult.

PROPOSITION 3. *Let L be any of the languages mentioned in Table 1. Then any two of* path containment, path unsatisfiability, *and* node unsatisfiability *are polynomially inter-reducible.*

The same holds for some other problems such as the *non-empty intersection* problem for path expressions [11]. Throughout this paper, we state our results in terms of *path containment*. However, many of the proofs make use of Proposition 3 and are formulated in terms of node satisfiability.

Each of the above problems can be relativised to a *Document Type Definition* (*DTD*). As in [1], we abstract away from DTD features such as default values and attributes, and define a DTD to be a triple (E, P, r), where (1) $E \subseteq \Sigma$ is a set of *element types*; (2) $r \in E$ is the *root type*; and (3) P is a function that assigns to each element of E a regular expression over E. A tree model M *conforms to* a DTD $D = (E, P, r)$ if the root of M is labeled with r, each node is labeled with a label from E, and for each node n with children $n_1, \ldots, n_k$, the word $Lab(n_1), \ldots, Lab(n_k)$ belongs to the language generated by $P(Lab(n))$.

The *path containment problem in the presence of DTDs* is as follows: given two path expressions α, β and a DTD D, is it true that for all tree models M conforming to D, $[\![\alpha]\!]^M_{\mathsf{PExpr}} \subseteq [\![\beta]\!]^M_{\mathsf{PExpr}}$? Path satisfiability and node satisfiability in the presence of DTDs are defined analogously. For each of these decision problems, the DTD-relativised version is at least as complex as the general version. For extensions of $\mathsf{CoreXPath}(*)$, it is not more complex either, as is implied by the following:

THEOREM 4 ([16]). *For each DTD D, there is a $\mathsf{CoreXPath}(*)$ node expression φ_D such that for all tree models M, M conforms to D iff the root of M satisfies φ_D.*

Thus, all upper bounds in Table 1 immediately generalize to the DTD-relativised case, except for the EXPSPACE-upper bound for $\mathsf{CoreXPath}_{\downarrow}(\cap)$, for which we will prove it by hand. In fact, in both cases, we could even handle *extended DTDs* (a formalism rich enough to capture the schema languages XSchema and RELAX NG) and *ancestor based patterns* [20, 15].

3. $\mathsf{CoreXPath}(*, \approx)$ IS IN EXPTIME

We show that path containment can be decided in EXPTIME for $\mathsf{CoreXPath}(*, \approx)$ using two way alternating tree automata.

To simplify the proof, we will work with a different, but equally expressive version of $\mathsf{CoreXPath}(*, \approx)$. This version differs in four aspects from the original one. First, we replace $\approx$ with the path operator loop, which tests whether a node is reachable from itself along a given path:

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

Note that $\mathsf{loop}(\alpha)$ can be expressed as $\alpha \approx .$ and, conversely, $\alpha \approx \beta$ can be written as $\mathsf{loop}(\alpha/\beta^\smile)$, where $\beta^\smile$ is the converse of β (defined inductively) [25]. Second, we drop the operator $\langle\alpha\rangle$. This can be done w.l.o.g. since $\langle\alpha\rangle$ can be expressed as $\mathsf{loop}(\alpha/\uparrow^*/\downarrow^*)$. Third, we replace the axes $\downarrow$ and $\uparrow$ with the *first-child* step ($\downarrow_1$) and its converse ($\uparrow_1$). Note that $\downarrow$ is equivalent to $\downarrow_1/\rightarrow^*$ and $\downarrow_1$ is equivalent to $\downarrow[\neg\langle\leftarrow\rangle]$. Likewise for $\uparrow$ and $\uparrow_1$. Fourth, we replace path expressions with NFAs (nondeterministic finite automata) whose alphabet is comprised of the

basic axes $\downarrow_1, \uparrow_1, \leftarrow, \rightarrow$ and node tests of the form $.[\varphi]$. This is justified by the observation that path expressions of $\mathsf{CoreXPath}(*, \approx)$ are just regular expressions over this alphabet if we replace subexpression of the form $\alpha[\varphi]$ with $\alpha/.[\varphi]$. It is well known that NFAs offer a succinct representation of such regular expressions.

To make things precise, we give a formal definition of the resulting version of XPath.

DEFINITION 5 ($\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$). *The node expressions and path automata of* $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ *are defined by simultaneous induction.*

► *Node expressions: Every label $p \in \Sigma$ is a node expression, and for every path automaton α, $\mathsf{loop}(\alpha)$ is a node expression. If φ and ψ are node expressions, then so are $\neg\varphi$ and $\varphi \wedge \psi$.*

► *Path automata: A path automaton is a tuple $\pi = (Q, \delta, q_0, q_f)$, where Q is a finite set of states, $\delta \subseteq^{fin} Q \times (\{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow\} \cup \{.[\psi] \mid \psi \text{ is a node expression}\}) \times Q$, and q_0 and q_f indicate the initial and final state.*

Note that since *skip* transitions can be defined as $.[\top]$, we can do with only one final state in the definition of NFAs. For a path automaton $\pi = (Q, \delta, q_0, q_f)$ and states $q, q' \in Q$, we will use $\pi_{(q,q')}$ as shorthand for (Q, δ, q, q'). The *length* of a path automaton is the number of states plus the total length of all node expressions φ occurring in transitions of the form $(q, .[\varphi], q')$.

Our aim is to translate $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ node expressions into two-way alternating tree automata (2ATAs). To prepare for this translation, we give an inductive characterization of loop.

LEMMA 6 (INDUCTIVE CHARACTERIZATION OF loop). *Let $\pi = (Q, \delta, q_0, q_f)$ be a $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ path automaton, and let $M = (N, R_\downarrow, R_\rightarrow, Lab)$ be any tree model. Define $\text{LOOPS}_\pi \subseteq N \times Q \times Q$ to be the smallest relation satisfying*

► $(n, q, q) \in \text{LOOPS}_\pi$ *for all $n \in N$ and $q \in Q$*

► *Whenever $(n, q_i, q_j) \in \text{LOOPS}_\pi$, $\delta(q_j, .[\varphi], q_k)$, and $n \in [\![\varphi]\!]^M_{\mathsf{NExpr}}$, then $(n, q_i, q_k) \in \text{LOOPS}_\pi$.*

► *Whenever $(n, q_j, q_k) \in \text{LOOPS}_\pi$, $\delta(q_i, .[\varphi], q_j)$ and $n \in [\![\varphi]\!]^M_{\mathsf{NExpr}}$, then $(n, q_i, q_k) \in \text{LOOPS}_\pi$.*

► *Whenever $nR_\tau m$, $(m, q_j, q_k) \in \text{LOOPS}_\pi$, $\delta(q_i, \tau, q_j)$, and $\delta(q_k, \bar{\tau}, q_\ell)$, then $(n, q_i, q_\ell) \in \text{LOOPS}_\pi$ (where $\tau \in \{\downarrow_1, \uparrow_1, \leftarrow, \rightarrow\}$ and $\bar{\tau}$ is the converse of τ)*

► *Whenever $(n, q_i, q_j) \in \text{LOOPS}_\pi$ and $(n, q_j, q_k) \in \text{LOOPS}_\pi$ then $(n, q_i, q_k) \in \text{LOOPS}_\pi$.*

Then $(n, q, q') \in \text{LOOPS}_\pi$ iff $n \models \mathsf{loop}(\pi_{q,q'})$.

PROOF SKETCH. The only difficult direction is the right-to-left direction. It is proved by induction on the length of the witnessing run of the NFA, using the crucial observation that, if the automaton starts at a node n and afterwards passes through another node m in the tree, in order to return to n it will have to pass through m again (this holds due to the fact that we chose $\downarrow_1, \uparrow_1, \leftarrow$ and $\rightarrow$ as our basic axes). □

Since we use a slightly non-standard version of 2ATAs, we give a brief introduction. Our 2ATAs work directly on tree models. They traverse such a model using the BASIC-STEPS $= \{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow, \epsilon\}$, 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 any node n of a tree model, POSSIBLE-STEPS(n) denotes the set of basic steps that can be performed from n. For each $a \in \text{POSSIBLE-STEPS}(n)$ we will denote by $n \cdot a$ the node reached from n by performing the basic step a.

DEFINITION 7 (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*

► *$\delta : (Q \times \Sigma \times \wp(\text{BASIC-STEPS})) \to \mathcal{B}^+(\text{BASIC-STEPS} \times Q)$ 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*

► *$Acc : Q \to \mathbb{N}$ specifies a "parity acceptance condition".*

Here, $\wp()$ denotes powerset and by $\mathcal{B}^+X$, we refer to the set of all positive Boolean formulas over variables from X, including true *and* false.

Note that the alphabet Σ of 2ATAs is the same as the alphabet underlying tree models (i.e., XML documents). Intuitively, $\delta(q, \sigma, A) = \psi$ means that if the automaton is in state q, reads σ, and the current node allows exactly the basic steps in A, then the transition is as described by ψ, in the usual sense of alternating automata. It follows from the results in [27] that for our version of 2ATAs, emptiness can be decided in EXPTIME. More details are given in Appendix A.

THEOREM 8. *Satisfiability of* $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ *node expressions is decidable in* EXPTIME.

PROOF. By reduction to the non-emptiness problem for 2ATAs. Let φ be any $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ node expression, and let $\varphi' = \downarrow^*/\varphi$. Note that φ is satisfiable iff φ' is satisfiable at the root of a tree model. Let $Cl(\varphi')$ be the smallest set of node expressions containing φ' such that

► $Cl(\varphi')$ is closed under taking subformulas and single negations

► For all $\mathsf{loop}(\pi) \in Cl(\varphi')$ and q_k, q_ℓ states of π, $\mathsf{loop}(\pi_{(q_k,q_\ell)})$ also belongs to $Cl(\varphi')$

Note that the size of $Cl(\varphi')$ and the length of its elements is bounded polynomially in the length of φ. We now define the 2ATA $A_{\varphi'}$ to be $(\Sigma, Q, \delta, q_0, Acc)$, where $Q = \{q_\psi \mid \psi \in Cl(\varphi')\}$, δ is defined as in Table 2, $q_0 = q_{\varphi'}$, Acc assigns 1 to all states of the form $q_{\mathsf{loop}(\alpha)}$, and 2 to all others (in other words, no state of the form $q_{\mathsf{loop}(\alpha)}$ may occur forever on a path in the run).

Using the inductive characterization of loop provided by Lemma 6, it is not hard to show that tree model belongs to $\mathcal{L}_{A_{\varphi'}}$ iff it satisfies φ' at the root. Thus, φ is satisfiable iff φ' is root-satisfiable iff $\mathcal{L}_{A_\varphi}$ is non-empty. □

By Proposition 3, together with the facts that (i) there is a linear translation from $\mathsf{CoreXPath}(*, \approx)$ to $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ and (ii) containment is already EXPTIME-hard for CoreXPath [1], we thus obtain

COROLLARY 9. *Containment for* $\mathsf{CoreXPath}(*, \approx)$ *is* EXPTIME*-complete.*

Together with known results about alternating automata [27], the proof of Theorem 8 also yields the following, which will come in handy later on.

THEOREM 10. *Every satisfiable* $\mathsf{CoreXPath}(*, \approx)$ *node expression φ is satisfied in a tree model of size $2^{O(|\varphi|)}$.*

Table 2: Transition function of the automaton $A_{\varphi'}$ used in the proof of Theorem 8

$$
\begin{array}{lcl}
\delta(\ell, q_p, \text{POSSIBLE-STEPS}) & = & \top \text{ if } \ell = p, \bot \text{ otherwise} \\
\delta(\ell, q_{\neg p}, \text{POSSIBLE-STEPS}) & = & \bot \text{ if } \ell = p, \top \text{ otherwise} \\
\\
\delta(\ell, q_{\psi\wedge\chi}, \text{POSSIBLE-STEPS}) & = & (\epsilon, q_\psi) \wedge (\epsilon, q_\chi) \\
\delta(\ell, q_{\neg(\psi\wedge\chi)}, \text{POSSIBLE-STEPS}) & = & (\epsilon, q_{\neg\psi}) \vee (\epsilon, q_{\neg\chi}) \\
\\
\delta(\ell, q_{\mathsf{loop}(\pi_{(q_i,q_j)})}, \text{POSSIBLE-STEPS}) & = & \top \text{ if } q_0 = q_f, \text{ otherwise} \\
 & & \bigvee_{\delta(q_i,\tau,q_k),\delta(q_\ell,\bar{\tau},q_j),\tau\in\text{POSSIBLE-STEPS}} (\tau, q_{\mathsf{loop}(\pi_{(q_k,q_\ell)})}) \\
 & & \vee \bigvee_{\delta(q_i,?\chi,q_k)} ((\epsilon,\chi) \wedge (\epsilon, q_{\mathsf{loop}(\pi_{(q_k,q_j)})})) \\
 & & \vee \bigvee_{\delta(q_k,?\chi,q_j)} ((\epsilon,\chi) \wedge (\epsilon, q_{\mathsf{loop}(\pi_{(q_i,q_k)})})) \\
 & & \vee \bigvee_{q_k\in Q} ((\epsilon, q_{\mathsf{loop}(\pi_{(q_i,q_k)})}) \wedge (\epsilon, q_{\mathsf{loop}(\pi_{(q_k,q_j)})})) \\
\\
\delta(\ell, q_{\neg\mathsf{loop}(\pi_{(q_i,q_j)})}, \text{POSSIBLE-STEPS}) & = & \bot \text{ if } q_0 = q_f, \text{ otherwise} \\
 & & \bigwedge_{\delta(q_i,\tau,q_k),\delta(q_\ell,\bar{\tau},q_j),\tau\in\text{POSSIBLE-STEPS}} (\tau, q_{\neg\mathsf{loop}(\pi_{(q_k,q_\ell)})}) \\
 & & \wedge \bigwedge_{\delta(q_i,?\chi,q_k)} ((\epsilon,\neg\chi) \vee (\epsilon, q_{\neg\mathsf{loop}(\pi_{(q_k,q_j)})})) \\
 & & \wedge \bigwedge_{\delta(q_k,?\chi,q_j)} ((\epsilon,\neg\chi) \vee (\epsilon, q_{\neg\mathsf{loop}(\pi_{(q_i,q_k)})})) \\
 & & \wedge \bigwedge_{q_k\in Q} ((\epsilon, q_{\neg\mathsf{loop}(\pi_{(q_i,q_k)})}) \vee (\epsilon, q_{\neg\mathsf{loop}(\pi_{(q_k,q_j)})}))
\end{array}
$$

4. CoreXPath$(*, \cap)$ IS IN 2-EXPTIME

We present exponential translations from CoreXPath$(\cap)$ to CoreXPath and from CoreXPath$(*, \cap)$ to CoreXPath$(*, \approx)$. From these translations, we derive upper bounds on the complexity and succinctness of CoreXPath$(\cap)$ and CoreXPath$(*, \cap)$.

THEOREM 11. *There is a single exponential translation from* CoreXPath$(\cap)$ *path expressions to* CoreXPath *path expressions.*

PROOF. The proof is along the same lines as in [2], where it is shown that positive CoreXPath$(\cap)$ path expressions can be translated to positive CoreXPath path expressions with an exponential blowup.

We will show how to do the translation for CoreXPath$(\cap)$ path expressions that do not contain $\cap$ inside qualifiers (here, by *qualifier* we mean a path modifier of the form $[\varphi]$). The general case follows by applying the translation repeatedly, in a bottom-up way.

Let α be a CoreXPath$(\cap)$ path expression not containing $\cap$ inside qualifiers. Introduce a unary predicate P_ψ for each node expression ψ that occurs in α as a test. Following [2] α can be translated in linear time to a positive existential first-order formula $\varphi(x, y)$ in the vocabulary consisting of $R_\downarrow$, $R_{\downarrow^*}$, $R_\rightarrow$ and $R_{\rightarrow^*}$ and the P_ψ's, such that α and $\varphi(x, y)$ define the same binary relation. Next, as shown in [1, 9], $\varphi(x, y)$ can be translated into a (positive) CoreXPath path expression β of exponential size, where the P_ψ's may occur in β as node tests. As a final step, we replace each P_ψ in β back by the original node expression ψ. □

The situation is less clear for CoreXPath$(*, \cap)$. In fact, we do not know whether it is possible to translate CoreXPath$(*, \cap)$ path expressions to CoreXPath$(*)$ at all. However, we do have a translation to CoreXPath$(*, \approx)$.

As in the previous section, it will be convenient to work with the equally expressive but more succinct language CoreXPath$_{\mathsf{NFA}}(*, \mathsf{loop})$.

LEMMA 12. *For all* CoreXPath$_{\mathsf{NFA}}(*, \mathsf{loop})$ *path automata α and β, there is a* CoreXPath$_{\mathsf{NFA}}(*, \mathsf{loop})$ *path expression γ of length $O(|\alpha| \cdot |\beta|)$ that is equivalent to $\alpha \cap \beta$.*

PROOF. Let $\alpha = (Q, \delta, q_0, q_f)$ and $\beta = (Q', \delta', q'_0, q'_f)$. Before we define γ, let us first provide the basic intuition. Suppose m and n are nodes in a tree model $M = (N, \ldots)$ such that $(m, n) \in [\![\alpha]\!]^M_{\mathsf{PExpr}}$ and $(m, n) \in [\![\beta]\!]^M_{\mathsf{PExpr}}$. There is a unique cycle-free path from m to n in the graph $(N, R_{\downarrow_1}, R_\rightarrow)$, and if we disregard loops, then every witnessing trace of α or β from m to n must travel exactly along this path in order to reach n from m.

This intuition drives our construction of γ below: we use loop to cut all loops made by α and β, so that the trip from m to n made by α and β can be performed synchronously. To be precise, we define γ as $(Q \times Q', \delta^\times, \langle q_0, q'_0\rangle, \langle q_f, q'_f\rangle)$, where the transition function $\delta^\times$ contains the following transitions:

- $\delta^\times(\langle q, q'\rangle, \tau, \langle r, r'\rangle)$ whenever $\delta(q, \tau, r)$ and $\delta'(q', \tau, r')$ (for $\tau \in \{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow\}$)
- $\delta^\times(\langle q, q'\rangle, .[\mathsf{loop}(\alpha_{(q,r)})], \langle r, q'\rangle)$
- $\delta^\times(\langle q, q'\rangle, .[\mathsf{loop}(\beta_{(q',r')})], \langle q, r'\rangle)$

It should be clear from the above discussion that γ defines exactly the intersection of the binary relations defined by α and β. □

THEOREM 13. *There is a single exponential translation from* CoreXPath$(*, \cap)$ *path expressions to* CoreXPath$(*, \mathsf{loop})$ *path expressions.*

PROOF. It follows from Lemma 12 that, for all CoreXPath$(*, \mathsf{loop})$ path expressions α and β, there is a CoreXPath$(*, \mathsf{loop})$ expression γ of length $2^{O(|\alpha|\cdot|\beta|)}$ that is equivalent to $\alpha \cap \beta$ (simply apply the usual exponential translation from NFAs to regular expressions). Applying this repeatedly, in a bottom-up fashion, we can translate any CoreXPath$(*, \cap)$ path expression into an exponentially long equivalent CoreXPath$(*, \mathsf{loop})$ path expression. □

COROLLARY 14. *Containment for* CoreXPath$(*, \cap)$ *is decidable in* 2-EXPTIME. *It is decidable in* EXPTIME *if there is a bound on the nesting depth of intersection.*

PROOF. The first half of the statement follows directly from Theorem 13. As for the second half, if there is a bound on the nesting depth of intersection, then Lemma 12 gives us a polynomial translation from CoreXPath$(*, \cap)$ to CoreXPath$_{\mathsf{NFA}}(*, \mathsf{loop})$. By Theorem 8, the latter is decidable in EXPTIME. □

A matching lower bound will be proved in Section 6. The above translations from $\mathsf{CoreXPath}(\cap)$ to $\mathsf{CoreXPath}$ and from $\mathsf{CoreXPath}(*, \cap)$ to $\mathsf{CoreXPath}(*, \approx)$ also allow to derive results about the relative succinctness of these languages. We will come back to this in Section 8.

5. $\mathsf{CoreXPath}_{\downarrow}(\cap)$ IS IN EXPSPACE

We will now show that containment can be decided in EXPSPACE for $\mathsf{CoreXPath}_{\downarrow}(\cap)$, the downward fragment of $\mathsf{CoreXPath}(\cap)$ (even if there is no bound on the nesting depth of intersection). As before, we will use Proposition 3, and prove the upper bound only for node satisfiability. Our proof is inspired by Ladner's PSPACE algorithm for testing satisfiability of modal formulas [13]. The heart of the argument lies in Lemma 16 below, which shows that we can restrict attention to tree models whose depth is bounded exponentially in the length of the input $\mathsf{CoreXPath}_{\downarrow}(\cap)$ expression.

As a first step, we show how we can get rid of union and intersection in path expressions, at the expense of an exponential blowup. We call a $\mathsf{CoreXPath}_{\downarrow}(\cap)$ path expression *simple* if it is of the form $\alpha_1/\cdots/\alpha_n$, where each α_i is of the form $\downarrow$, $\downarrow^*$ or $.[\varphi]$. We use $|\alpha|$ to denote the length of a path expression α (the number of symbols needed to write it), and likewise for node expressions.

LEMMA 15. *For every* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *path expression* α*, there is a set of simple* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *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.

The proof is based on a careful case distinction. Details can be found in Appendix B.

For any simple path expression $\alpha = \alpha_1/\cdots/\alpha_n$, we will use $\mathsf{suff}(\alpha)$ to denote the set $\{\alpha_i/\cdots/\alpha_n \mid 1 \leq i \leq n\}$. For any node expression φ, $\mathsf{sub}(\varphi)$ is the set of all sub-expressions of φ, i.e., node expressions that occur as a subterm of φ (possibly nested inside a path sub-expression occurring in φ). Finally, $\mathsf{aux}(\varphi)$ is the set of all node expressions of the form $\langle\beta\rangle$ with $\beta \in \mathsf{suff}(\mathsf{inst}(\alpha))$ for some $\langle\alpha\rangle \in \mathsf{sub}(\varphi)$. Intuitively, $\mathsf{sub}(\varphi) \cup \mathsf{aux}(\varphi)$ contains all those node expressions that might be needed in order to compute inductively the set of nodes that satisfy φ.

LEMMA 16. *Every satisfiable* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *node expression* φ_0 *is satisfiable in a tree of height* $2^{O(|\varphi_0|^3)}$.

The proof of this lemma consists of a slightly intricate manipulation of tree models. The basic idea is to describe each node n in a tree model by a *type* which includes (a subset of) the node expressions from $\mathsf{sub}(\varphi) \cup \mathsf{aux}(\varphi)$ true at n. Although there are doubly exponentially many such types, a careful counting argument shows that only singly exponentially many types can be realized on each individual branch of a tree. Moreover, if a branch contains two nodes that have the same type, the model can be contracted into a smaller one. Details are given in Appendix B.

To present the decision procedure, we need a few further preliminaries. For any $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression φ, let $\mathsf{cl}(\varphi) = \{\psi, \neg\psi \mid \psi \in \mathsf{sub}(\varphi) \cup \mathsf{aux}(\varphi)\}$.

```
guess a complete φ0-type t;
if φ0 ∉ t then return 'No' else return check(t,0);

procedure check(t, i):
  if i > 2^p(|φ0|) then return 'No' else continue;

  for each demand φ ∈ t do
    let the remainder of φ be φ';
    guess a complete φ0-type t';
    if φ' ∉ t' or t ↛ t' or check(t', i+1)='No' then
        return 'No' else continue;

  return 'Yes';
```

Figure 2: Non-deterministic exponential space algorithm for testing satisfiability of a $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression φ_0.

DEFINITION 17 (COMPLETE TYPE). *Let* φ_0 *be any* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *node expression. A* complete φ_0-type *is any* $t \subseteq \mathsf{cl}(\varphi_0)$ *satisfying the following conditions:*

- t *contains at most one label from* Σ *as an element.*
- *For all* $\neg\varphi \in \mathsf{cl}(\varphi)$, $\varphi \in t$ *iff* $\neg\varphi \notin t$.
- *For all* $(\varphi \wedge \psi) \in \mathsf{cl}(\varphi_0)$, $(\varphi \wedge \psi) \in t$ *iff* $\varphi \in t$ *and* $\psi \in t$.
- *For all* $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$, $\langle\alpha\rangle \in t$ *iff there is a* $\beta \in \mathsf{inst}(\alpha)$ *such that* $\langle\beta\rangle \in t$.
- *For all* $\langle.[\psi]/\beta\rangle \in \mathsf{aux}(\varphi_0)$, $\langle.[\psi]/\beta\rangle \in t$ *iff* $\psi \in t$ *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 φ_0-type. 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 φ_0-types, t, t', 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 type of a child of a node with type t.

The (non-deterministic) algorithm for node satisfiability in $\mathsf{CoreXPath}_{\downarrow}(\cap)$ is given in Figure 2. It takes as input a node expression φ_0, and it tries to recursively construct a model for the expression, while keeping in memory at any point only a single branch of length at most $2^{p(|\varphi_0|)}$, where p is the precise polynomial of Lemma 16. The correctness of the algorithm is proved in Appendix B. Since NEXPSPACE = EXPSPACE by Savitch's theorem, we obtain that $\mathsf{CoreXPath}_{\downarrow}(\cap)$ is in EXPSPACE.

The above construction can be extended to the case with DTDs, giving us an EXPSPACE algorithm for testing containment for $\mathsf{CoreXPath}_{\downarrow}(\cap)$ in the presence of DTDs. In the case relative to a DTD $D = (E, P, r)$, the check procedure, when called with input type t, has to guess not just a type for each demand in t, but also a word belonging to $P(a)$, where a is the unique label belonging to t. Extended DTDs and ancestor based patterns can be handled as well.

THEOREM 18. *Containment for* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *in the presence of DTDs is decidable in* EXPSPACE.

A matching lower bound will be proved in the next section.

6. LOWER BOUNDS FOR $\mathsf{CoreXPath}(*, \cap)$

We now provide matching lower bounds for the upper bounds from the previous sections: 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 any easy way.

As usual, we concentrate on satisfiability of node expressions. By Proposition 3, the results carry over to containment for path expressions. It is convenient to work with a generalization of tree models in which nodes can satisfy more than one label, i.e., where the labelling function Lab of models is a function from N to 2^{Σ}. We call such models *tree models with multi-labels*. The following lemma shows that, in proving our lower bounds, we can safely use tree-models with multi-labels.

LEMMA 19. *Satisfiability of* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *node expressions on tree-models with multi-labels can be reduced in polynomial time to satisfiability of* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *node expressions on standard tree models. The same holds for* $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$, $\mathsf{CoreXPath}_{\downarrow\uparrow}(\cap)$ *and* $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$.

The lemma is proved via an easy encoding of the multi-labels of a node n through the (standard) labels of additional children of n, similar to what is done in [8]. Details are given in Appendix C. In the remainder of this section, we only consider tree models with multi-labels. Our first aim is to prove the following result.

THEOREM 20. *Containment for* $\mathsf{CoreXPath}_{\downarrow,\uparrow}(\cap)$ *is 2-*EXPTIME*-hard.*

The proof is by reduction of the word problem for exponentially space bounded alternating Turing machines (ATMs). Recall that 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 Γ the *work alphabet* containing a *blank symbol* $\Box$ and satisfying $\Lambda \subseteq \Gamma$; $q_0 \in Q_{\exists} \cup Q_{\forall}$ 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 will write $\Delta(q, a)$ for $\{(q', b, M) \mid (q, a, q', b, M) \in \Delta\}$.

There exists an exponentially space bounded ATM $\mathcal{M} = (Q, \Lambda, \Gamma, q_0, \Delta)$ whose word problem is 2-EXPTIME-hard [3]. 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 w.l.o.g. that M never attempts to move left on the left-most 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 $w \in L(\mathcal{M})$ iff $\varphi_{\mathcal{M},w}$ is satisfiable.

In models of $\varphi_{\mathcal{M},w}$, nodes are used to represent tape cells and subtrees are used to represent configurations. This is illustrated in Figure 3, in which the triangles denote *configuration trees*, i.e., binary trees of depth k whose leaves encode a configuration. The figure shows a configuration (left tree) with two successor configurations (middle and right tree). The r and $\neg r$ labels are used as markers, as explained later on. In the reduction, we use the set of labels $Q \cup \Gamma \cup \{r, c_0, \ldots, c_{k-1}\}$, whose intuitive meaning is as follows (recall that we work with multi-label trees):

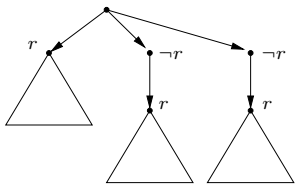


Figure 3: Successor configurations in $\mathsf{CoreXPath}_{\downarrow,\uparrow}(\cap)$.

► $q \in Q$ is true in a leaf n of a configuration tree 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 leaf n of a configuration tree if a is the symbol on the tape cell represented by n;

► r is used to identify the roots of configuration trees and is important for "travelling" between successor configurations using path expressions;

► $c_0, \ldots, c_{k-1}$ describe a counter C in binary coding for counting the 2^k tape cells of configurations, with the leftmost cell having counter value $C = 0$;

In the following, we use $\{\alpha \,!\, \varphi\}$ as an abbreviation for $\neg\langle\alpha[\neg\varphi]\rangle$, i.e., the semantics is

$$[\![\{\alpha \,!\, \varphi\}]\!]_{\mathsf{NExpr}} = \{n \in N \mid \forall m \in N.\, (n, m) \in [\![\alpha]\!]_{\mathsf{PExpr}} \text{ implies } m \in [\![\varphi]\!]_{\mathsf{NExpr}}\}.$$

This abbreviation corresponds to the *box operator* of modal logic. 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. For all $i < k$, define:

$$\varphi_{\mathsf{tree}} := \bigwedge_{i<k} \{\downarrow^*[r]/\downarrow^i \,!\, (\langle\downarrow[c_i]\rangle \wedge \langle\downarrow[\neg c_i]\rangle \wedge \bigwedge_{j<i}((c_j \Rightarrow \{\downarrow \,!\, c_j\}) \wedge (\neg c_j \Rightarrow \{\downarrow \,!\, \neg c_j\})))\}$$

where $\downarrow^i$ denotes the i-fold composition $\downarrow/\cdots/\downarrow$. Since we are only allowed to travel up and down, we cannot ensure that every value of C occurs *at most* once among the leaves. Instead, we ensure that cells with identical C values are labelled in the same way. Define for all $a, b \in \Gamma$ and all $i < k$:

$$\alpha_{ab} := .[a]/\uparrow^k/\downarrow^k[b]$$
$$\alpha_{=i} := (.[c_i]/\uparrow^k/\downarrow^k[c_i]) \cup (.[\neg c_i]/\uparrow^k/\downarrow^k[\neg c_i])$$

Now, the path expression $\alpha_{=\mathsf{cur}}$ travels between any two leafs of a configuration tree with the same C value and φ_{uni} ensures unique labels.

$$\alpha_{=\mathsf{cur}} := \bigcap_{i<k} \alpha_{=i}$$
$$\varphi_{\mathsf{uni}} := \bigwedge_{a,b\in\Gamma, a\neq b} \neg\langle\downarrow^*[r]/\downarrow^k/(\alpha_{ab} \cap \alpha_{=\mathsf{cur}})\rangle$$

It is easy to construct a node expression φ_{tape} which ensures that (i) every tape cell is marked with at least (and thus exactly) one symbol from Γ and never with two different states, (ii) each configuration has at most one cell marked with the tape head (which involves a path expression $\alpha_{\neq\mathsf{cur}}$ similar to $\alpha_{=\mathsf{cur}}$ above), and (iii) the initial configuration is such that the head is on the left-most tape cell, the ATM is in state q_0, and the tape is labelled with the input word followed by blanks. Details are left to the reader. We now say that cells not underneath the head are labelled with the same alphabet symbol in the consecutive configuration. A central role is played by the path expression $\alpha_{=\mathsf{nxt}}$, which travels from leafs of a

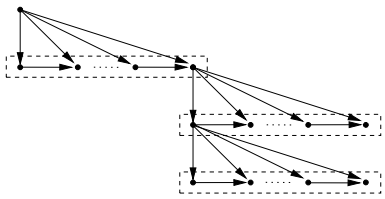


Figure 4: Successor configurations in $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$.

configuration to leafs of the successor configurations that represent the same tape cell.

$$\begin{aligned}\alpha_{=\mathsf{nxt}} &:= \bigcap_{i<k} \big((\downarrow/\uparrow[c_i]/\uparrow^{k+1}/\downarrow[\neg r]/\downarrow[r]/\downarrow^k[c_i]) \cup \\ &\qquad (\downarrow/\uparrow[\neg c_i]/\uparrow^{k+1}/\downarrow[\neg r]/\downarrow[r]/\downarrow^k[\neg c_i])\big)\\ \varphi_{\mathsf{id}} &:= \bigwedge_{a\in\Gamma} \big(\,(\neg h \wedge a) \Rightarrow \{\alpha_{=\mathsf{nxt}}\,!\,a\}\,\big)\end{aligned}$$

Observe that we use the additional root labelled $\neg r$ of successor configurations (c.f. Figure 3) to distinguish them from the current configuration.

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}:

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

where $\alpha_{\neq i}$ is defined analogously to $\alpha_{=i}$. Now, the following node expression takes care of proper transitions:

$$\begin{aligned}\varphi_\Delta := \big\{&\downarrow^*[r]/\downarrow^k !\\ &\bigwedge_{q\in Q_\exists, a\in\Gamma} \big(\,q\wedge a \Rightarrow \bigvee_{(p,b,M)\in\Delta(q,a)} \langle \alpha_{=\mathsf{nxt}}[b \wedge \{\alpha_{M\mathsf{cur}}\,!\; p\}]\rangle\,\big) \wedge\\ &\bigwedge_{q\in Q_\forall, a\in\Gamma} \big(\,q\wedge a \Rightarrow \bigwedge_{(p,b,M)\in\Delta(q,a)} \langle \alpha_{=\mathsf{nxt}}[b \wedge \{\alpha_{M\mathsf{cur}}\,!\; p\}]\rangle\,\big)\,\big\}\end{aligned}$$

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: $\varphi_{\mathsf{acc}} := [\downarrow^*]\neg q_r$. Altogether, the machine's behaviour is described by the node expression

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

It is not hard to show that $w \in L(\mathcal{M})$ iff $\varphi_{\mathcal{M},w}$ is satisfiable, which establishes Theorem 20.

The following result is proved similarly, again by a reduction of the word problem of exponentially space-bounded ATMs.

THEOREM 21. *Containment for* $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$ *is 2-*EXPTIME*-hard.*

We only illustrate the most important ideas of the reduction. It uses the same labels as in the previous reduction, except that we drop r. In $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$, configurations have to be represented in a different way than before. This is illustrated in Figure 4, in which the horizontal sequences are of length 2^k and represent a configuration. As before, the figure shows one configuration (left box) together with two successor configurations (right boxes). We can access neighboring cells in the same configuration by travelling to the right, and we can access cells in successor configurations by means of the path expression $\rightarrow^*[\neg\langle\rightarrow\rangle]/\downarrow^*/\rightarrow^*$.

All relevant properties can in fact be described in $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$ without using the one-step sibling axes $\rightarrow$ (which is not present in the original version of CoreXPath). To ensure that each horizontal sequence is of the appropriate length, we use the counter C, enforcing that (i) every non-leaf has a child with counter value 0; (ii) each node with counter value $x < 2^k$ sees via $\rightarrow^*$ a node with counter value $x+1$; and (iii) there is no node that sees via $\rightarrow^*$ a node with the same counter value. To achieve (i)-(iii), we use the following node expression:

$$\begin{aligned}\varphi_{inc} := \{\downarrow^*!\; &\big(\langle\downarrow\rangle \Rightarrow \langle\downarrow[\neg c_0 \wedge \cdots \wedge \neg c_{k-1}]\rangle\big) \wedge\\ &\big(\neg c_0 \vee \cdots \vee \neg c_{k-1} \Rightarrow \langle\alpha_{inc} \cap \rightarrow^*\rangle\big) \wedge\\ &\neg\langle\alpha_= \cap \rightarrow^*\rangle\;\}\\ \alpha_{inc} := \bigcup_{i<k} \Big(&.[c_0 \wedge \cdots \wedge c_{i-1} \wedge \neg c_i]\,/\\ &\bigcap_{i<j<k} \big((c_j/\rightarrow^*/c_j) \cup (\neg c_j/\rightarrow^*/\neg c_j)\big)\,/\\ &.[\neg c_1 \wedge \cdots \wedge \neg c_{i-1} \wedge c_i]\Big)\end{aligned}$$

It is straightforward to adapt the node expressions φ_{tape}, φ_{id}, φ_Δ, and φ_{acc} to the new setup. The expressions φ_{tree} and φ_{uni} are not needed.

Adding transitive closure to $\mathsf{CoreXPath}_{\downarrow}(\cap)$ also makes containment 2-EXPTIME. This in fact follows from a known result. $\mathsf{CoreXPath}_{\downarrow}(*,\cap)$ can be seen as a notational variant of *propositional dynamic logic with intersection* (*IPDL*). It was proved in [14] that satisfiability of IPDL formulas in finite tree models (with multi-labels) is 2-EXPTIME-hard. Thus, we obtain

THEOREM 22. *Containment for* $\mathsf{CoreXPath}_{\downarrow}(*,\cap)$ *is* 2-EXPTIME*-hard.*

Finally, we consider the *downwards fragment* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ and provide a matching EXPSPACE lower bound for Theorem 18.

THEOREM 23. *Containment for* $\mathsf{CoreXPath}_{\downarrow}(\cap)$ *is* EXPSPACE*-hard.*

The proof is by a reduction of the word problem of exponentially *time*-bounded ATMs. According to [3], 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 M never attempts to move left on the left-most 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 $w \in L(\mathcal{M})$ iff $\varphi_{\mathcal{M},w}$ is satisfiable.

Figure 5 shows how we represent a configuration and two successor configurations. Each box encloses a configuration, which is represented by a downward sequence of length 2^k. The two lower boxes represent the successor configuration of the upper box. We use the labels from the previous reductions and introduce additional labels $d_0, \ldots, d_{k-1}$ for implementing a second counter D. While the purpose of C is still to count the tape cells of a configuration, the purpose of D is to count successive configurations. Both counters are initialized with value 0. With each child-step, C is incremented modulo 2^k. If the value of C changes from $2^k - 1$ to 0, the counter D is incremented as well, otherwise it stays the same.

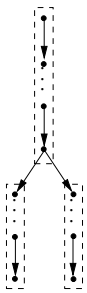


Figure 5: Successor configurations in $\mathsf{CoreXPath}_{\downarrow}(\cap)$.

For φ_{id} and φ_{Δ}, we need a path expression that accesses the corresponding tape cell in successor configurations, i.e., the node whose C value is the same as the current C value, and whose D value is the current D value plus one. Such a path expression is easy to find, by analogy to $\alpha_{=\mathsf{cur}}$ and α_{Rcur} above.

7. PATH COMPLEMENTATION AND FOR-LOOPS

We now consider two extensions of CoreXPath for which path containment is non-elementary. Recall that a decision problem is called *non-elementary* if the time needed to solve it cannot be bounded by any exponential tower of constant height.

First, we consider $\mathsf{CoreXPath}(-)$, the extension of CoreXPath with path complementation. Path containment is non-elementary even for a small fragment L of $\mathsf{CoreXPath}(-)$, in which path expressions are formed as follows:

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

for $p \in \Sigma$ and with $\downarrow^+$ abbreviating $\downarrow/\downarrow^*$. Note that L has only downward axes, lacks complex node tests, and lacks union as a primitive operator.

THEOREM 24. *Containment for L is non-elementary.*

PROOF. We now give a reduction from the non-emptiness problem for *star-free expressions* $r := a \mid (rr') \mid (r \cup r') \mid -r$, which is well-known to be non-elementary [24]. First, note that using the path complementation operator, we can define path intersection and union:

$$\begin{array}{lcl} \alpha \cap \beta & \equiv & (\alpha - (\alpha - \beta)) \\ \alpha \cup \beta & \equiv & \downarrow^+ - ((\downarrow^+ - \alpha) \cap (\downarrow^+ - \beta)) \end{array}$$

The exponential blowup involved in this definition of union is of no importance, as our intention is merely to show non-elementarity. Also, for the latter equivalence, note that all path expressions of L define a subrelation of $\downarrow^+$.

Star-free expressions can now be translated into L:

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

It can be shown by induction on r that, for any tree model M with nodes n, m, and for any star-free expression r, $(n,m) \in [\![\mathsf{tr}(r)]\!]^M_{\mathsf{PExpr}}$ iff 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 $(Lab(n_2), \ldots, Lab(n_k))$ belongs to the language defined by r. It follows that r defines a non-empty language iff $\mathsf{tr}(r)$ is not contained in $\downarrow^+ - \downarrow^+$. □

Similarly, satisfiability is non-elementary for L. This improves on a result from [12], which shows that L-satisfiability is NP-hard.

Next, let us consider $\mathsf{CoreXPath}(\mathsf{for})$. The for-construct in XPath 2.0, allows iteration over a node set, using a bound variable. Formally, $\mathsf{CoreXPath}(\mathsf{for})$ is obtained by introducing a countably infinite set of node variables \$i, \$j, …, 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*.

► We allow node tests of the form ". is $\$i$", interpreted as follows:

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

► We allow path expressions of the form "for $\$i$ inα return β", interpreted as follows:

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

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 $\mathsf{CoreXPath}(\mathsf{for})$ is denoted by $\mathsf{CoreXPath}_{\downarrow}(\mathsf{for})$.

THEOREM 25. *Containment for $\mathsf{CoreXPath}_{\downarrow}(\mathsf{for})$ is non-elementary, even if only one variable is used in the expressions.*

PROOF. Using a single variable, we can express complementation: if α, β are downward path expressions, then $\alpha - \beta$ is equivalent to for $\$i$ in α return.$[\neg\langle\beta[\text{. is } \$i]\rangle]/\downarrow^*[\text{. is } \$i]$. It follows that containment for $\mathsf{CoreXPath}_{\downarrow}(\mathsf{for})$ is at least as complex as for $\mathsf{CoreXPath}_{\downarrow}(-)$, i.e., non-elementary. □

8. SUCCINCTNESS

We have seen in Section 4 that there are exponential translations from $\mathsf{CoreXPath}(\cap)$ to CoreXPath and from $\mathsf{CoreXPath}(*,\cap)$ to $\mathsf{CoreXPath}(*,\approx)$. This shows that $\mathsf{CoreXPath}(\cap)$ and $\mathsf{CoreXPath}(*,\cap)$ are at most exponentially more succinct than CoreXPath and $\mathsf{CoreXPath}(*,\approx)$, respectively. They are indeed *precisely* exponentially more succinct.

Suppose two languages, L and L', have the same expressive power. We say that L is *exponentially* (or, *non-elementarily*) *more succinct* than L', if there is a sequence 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 at least exponentially (non-elementarily) in i.

THEOREM 26.

- $\mathsf{CoreXPath}(\cap)$ *is exponentially more succinct than* $\mathsf{CoreXPath}(\approx)$.
- $\mathsf{CoreXPath}(*,\cap)$ *is exponentially more succinct than* $\mathsf{CoreXPath}(*,\approx)$.

PROOF. We may w.l.o.g. restrict our attention to any subset of the set of all finite tree models. Let $\mathcal{T}^1_{p,q}$ be the class of tree models in which each node has at most one child, and each node is labelled 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 i-th 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 tag name:

For all $i, j \leq n - 2k$, if $u_i, u_{i+1}, u_j, u_{j+1}$ all have tag name 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 nodd expression of $\mathsf{CoreXPath}(\cap)$ *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 $\bar{\alpha_\ell}$ 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 hold 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

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

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

We already saw that every $\mathsf{CoreXPath}_{\mathsf{NFA}}(*, \mathsf{loop})$ node expression can be translated in polynomial time to 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 [27]. Now, Etessami, Vardi and Wilke [5] proved that any automaton of the latter kind defining φ_k has at least 2^{2^k} many states. The claim follows. □

THEOREM 27. $\mathsf{CoreXPath}(*, -)$ *is non-elementarily more succinct than* $\mathsf{CoreXPath}(*, \cap)$

PROOF. It follows from Theorem 24 that the size of the smallest tree model for a satisfiable $\mathsf{CoreXPath}(-)$ node expression φ cannot be bounded elementarily in the length of φ (indeed, if there were an elementary bound, then this would easily yield an elementary decision procedure for testing $\mathsf{CoreXPath}(-)$ satisfiability: try all models). This means that there is a sequence of $\mathsf{CoreXPath}(-)$ formulas $(\varphi_i)_{i\in\mathbb{N}}$ of length linear in i and such that each φ is satisfiable but such that the smallest tree model for φ_i cannot be bounded by an elementary function of i. On the other hand, we know from Theorem 10 and Theorem 13 that every satisfiable $\mathsf{CoreXPath}(*, \cap)$ expression φ is satisfied in a tree model of size doubly exponential in the length of φ. Therefore, any sequence $(\varphi'_i)_{i\in\mathbb{N}}$ of $\mathsf{CoreXPath}(*, \cap)$ formulas such that each φ'_i is equivalent to φ_i must grow non-elementarily in length. □

THEOREM 28. $\mathsf{CoreXPath}(\mathsf{for})$ *is (at least) exponentially more succinct than* $\mathsf{CoreXPath}(-)$.

PROOF. Let FO be the first-order language, interpreted on tree models, 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 $\mathsf{CoreXPath}(-)$ into the three variable fragment of FO, and there is a linear translation from full FO into $\mathsf{CoreXPath}(\mathsf{for})$. It was shown in [10] that, on tree models, FO is exponentially more succinct than its three variable fragment (in fact, the proof does not even involve unary predicates). It follows that $\mathsf{CoreXPath}(\mathsf{for})$ is also exponentially more succinct than $\mathsf{CoreXPath}(-)$. □

Acknowledgements

We would like to thank Maarten Marx for his useful comments. The first author was supported by NWO research grant 639.021.508, and the second author was supported by the EU funded IST-2005-7603 FET Project Thinking Ontologies (TONES).

9. REFERENCES

[1] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *Proc. of PODS 2005*, pages 25–36. ACM Press, 2005.

[2] M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. *Theoretical Computer Science*, 336(1):3–31, 2005.

[3] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[4] A. Deutsch and V. Tannen. Containment and integrity constraints for XPath fragments. In *Proc. of KRDB 2001*, volume 45 of *CEUR Workshop Series*. 2001.

[5] K. Etessami, M. Y. Vardi, and T. Wilke. First order logic with two variables and unary temporal logic. *Information and computation*, 179(2):279–295, 2002.

[6] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Rewriting regular XPath queries on XML views. In *Proc. of ICDE 2007*, 2007.

[7] G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proc. of LICS 2002*, pages 189–202. IEEE Computer Society, 2002.

[8] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of XPath query evaluation and XML typing. *Journal of the ACM*, 52(2):284–335, 2005.

[9] G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. In *Proc. of PODS 2004*, pages 189–200. ACM Press, 2004.

[10] M. Grohe and N. Schweikardt. The succinctness of first-order logic on linear orders. *Logical Methods in Computer Science*, 1(1), 2005.

[11] B. C. Hammerschmidt, M. Kempa, and V. Linnemann. On the intersection of XPath expressions. In *Proc. of IDEAS 2005*, 2005.

[12] J. Hidders. Satisfiability of XPath expressions. In *Proc. of DBPL*, pages 21–36, 2003.

[13] R. E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6(3):467–480, 1977.

[14] M. Lange and C. Lutz. 2-ExpTime lower bounds for propositional dynamic logics with intersection. *Journal of Symbolic Logic*, 70(5):1072–1086, 2005.

[15] W. Martens, F. Neven, T. Schwentick, and G. Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems* 31(3):770–813, 2006.

[16] M. Marx. XPath with conditional axis relations. In *Proc. of EDBT 2004*, volume 2992 of *LNCS*. Springer, 2004.

[17] M. Marx. Conditional XPath. *ACM Transactions on Database Systems*, 30(4):929–959, 2005.

[18] M. Marx and M. de Rijke. Semantic characterizations of navigational XPath. *ACM SIGMOD Record*, 34(2):41–46, 2005.

[19] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. of PODS 2002*, pages 65–76. ACM Press, 2002.

[20] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology* 5(4):660–704, 2005.

[21] F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs and variables. *Logical Methods in Computer Science*, 2:1–30, 2006.

[22] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. In *Proc. of XMLDM 2002*, pages 109–127. Springer Verlag, 2002.

[23] T. Schwentick. XPath query containment. *ACM SIGMOD Record*, 33(1):101–109, 2004.

[24] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory*. PhD thesis, Department of Electrical Engineering, MIT, 1974.

[25] B. ten Cate. The expressivity of XPath with transitive closure. In *Proc. of PODS 2006*, pages 328–337. ACM Press, 2006.

[26] B. ten Cate and M. Marx. Axiomatizing the logical core of XPath 2.0. In *Proc. of ICDT 2007*, 2007.

[27] M. Vardi. Reasoning about the past with two-way automata. In *Proc. of ICALP 1998*, volume 1443 of *LNCS*, pages 628–641. Springer, 1998.

[28] P. T. Wood. Containment for XPath fragments under DTD constraints. In *Proc. of ICDT 2003*, pages 300–314, 2003.

APPENDIX

A. ALTERNATING TREE AUTOMATA

We give more details regarding the two-way alternating tree automata used in Section 3. Recall that BASIC-STEPS $= \{\downarrow_1, \uparrow_1, \rightarrow, \leftarrow, \epsilon\}$ are the possible moves of the automaton, and that for any node n of a tree model, POSSIBLE-STEPS(n) denotes the set of basic steps that can be performed at n. Also recall that $n \cdot a$ denotes the node reached from n by performing the basic step a. For convenience, we repeat the definition of 2ATAs.

DEFINITION 7 (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*
- *$\delta : (Q \times \Sigma \times \wp(\text{BASIC-STEPS}) \to \mathcal{B}^+(\text{BASIC-STEPS} \times Q)$ 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*
- *$Acc : Q \to \mathbb{N}$ specifies a "parity acceptance condition".*

Here, by $\mathcal{B}^+X$, we refer to the set of all positive Boolean formulas over variables from X (including true *and* false*).*

DEFINITION 29 (ACCEPTING RUNS). *An* accepting run *of a 2ATA $A = (Q, \delta, q_0, Acc)$ on a tree model $M = (N, R_\downarrow, R_\rightarrow, Lab)$ is a tree whose nodes are labelled with pairs $(n, q) \in N \times Q$, such that the following conditions hold:*

- ***(Initial configuration)***
  The root of the run is labelled by $(root_T, q_0)$.
- ***(Respecting the transition function)***
  If a node x in the run is labelled by (n, q), and $\delta(q, Lab(n), \text{POSSIBLE-STEPS}(n)) = \theta$, then there is a set $S \subseteq (\text{POSSIBLE-STEPS}(n) \times Q)$ making θ true, such that for each $(a, q) \in S$, x has a child labelled $(n \cdot a, q)$.
- ***(Acceptance condition)***
  For each infinite path in the run, the lowest number assigned by Acc to a state occurring infinitely often on that path is even*.*

A accepts *a tree model M if it has an accepting run on M. $\mathcal{L}_A$ is the set of tree models accepted by A.*

THEOREM 30 (EMPTINESS IS IN EXPTIME). *Given a 2ATA A, it is decidable in exponential time whether $\mathcal{L}_A$ is empty.*

PROOF. We give a polynomial reduction of the emptiness problem for our version of 2ATAs to the emptiness problem of standard 2ATAs on infinite binary ranked trees with parity acceptance condition. The latter is well-known to be EXPTIME-complete [27].

For a tree model M, define the *decoration* of M to be the tree model M^d whose node labels are from the set $\Sigma^d := \Sigma \times 2^{\text{BASIC-STEPS}}$ such that for all nodes n, the second component of $Lab'(n)$ is POSSIBLE-STEPS(n). For a set $\mathcal{L}$ of tree models, $\mathcal{L}^d$ is the set of decorations of trees in $\mathcal{L}$.

We can view the elements of $\mathcal{L}^d$ as a special case of binary trees with two partial successor functions f_1, f_2, corresponding to the *first child* and *next sibling* relations. These binary trees have the special properties that (i) they are finite, and (ii) the root has no f_2-successor (i.e., it has no siblings).

From this perspective, we can view a 2ATA on tree models as a 2ATA on binary trees over the alphabet Σ^d. By intersection with a 2ATA that ensures (i), (ii), and a proper labelling in the second component, we obtain a 2ATA on binary trees accepting $\mathcal{L}^d$ (intersection is trivial for alternating automata). Clearly, $\mathcal{L}^d$ is empty iff $\mathcal{L}$ is, and thus we obtain the desired reduction. □

B. MISSING PROOFS

LEMMA 15. *For every* $\mathsf{CoreXPath}_\downarrow(\cap)$ *path expression α, there is a set of simple* $\mathsf{CoreXPath}_\downarrow(\cap)$ *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, we will use ε to denote a simple path expression of length zero (i.e., a concatenation of zero steps), and we treat α/ε as identical to α. For α and β simple path expressions, $\mathsf{int}\{\alpha, \beta\}$ is defined inductively, as follows.

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

An inductive argument shows that $\bigcup \mathsf{int}\{\alpha, \beta\}$ is equivalent to $\alpha \cap \beta$, $|\mathsf{int}\{\alpha, \beta\}|$ is $2^{O(|\alpha|+|\beta|)}$, and each $\gamma \in \mathsf{int}\{\alpha, \beta\}$ satisfies $|\gamma| \leq |\alpha| + |\beta|$.

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

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

An inductive argument shows that $\bigcup \mathsf{inst}(\alpha)$ is equivalent to α, $|\mathsf{inst}(\alpha)|$ is $2^{O(|\varphi|^2)}$ and each $\gamma \in \mathsf{inst}(\alpha)$ satisfies $|\gamma| \leq 4 \cdot |\alpha|$. Moreover, all node expressions occurring in expressions from $\mathsf{inst}(\alpha)$ occur in α. □

LEMMA 16. *Every satisfiable* $\mathsf{CoreXPath}_\downarrow(\cap)$ *node expression φ_0 is satisfiable in a tree of height* $2^{O(|\varphi_0|^3)}$.

PROOF. The basic idea of the proof is that every sufficiently long branch in a model satisfying φ_0 must contain two nodes, say $nR_{\downarrow^*}m$, that are of the same "type". Then, we can shorten the path by replacing the subtree rooted by n with the subtree rooted by 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 witnessing α-path.

We must first define some auxiliary notions. A *witness function* for a tree model M will be a function π assigning to each node n a set of node expressions that are true at n, such that the following conditions hold for all nodes n:

- If n satisfies some $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$, then $\pi(n)$ includes a formula $\langle\beta\rangle$ with $\beta \in \mathsf{inst}(\alpha)$.
- For all $\langle.[\varphi]/\beta\rangle \in \pi(n)$, also $\langle\beta\rangle \in \pi(n)$.
- 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 $\langle\beta'\rangle \in \pi(n')$.

A witness function can be constructed for any model M as follows: start with the function that assigns to each node the empty set, and as long as one of the three above conditions is not satisfied, keep extending the sets as required. For instance, if a node n satisfies $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$ but there is no $\beta \in \mathsf{inst}(\alpha)$ for which $\langle\beta\rangle \in \pi(n)$, then pick any such $\langle\beta\rangle$ true at n (which exists by the semantics) and add it to $\pi(n)$. Any witness function for M constructed in this way will be called a *minimal witness function*, as it contains no more information than needed.

Fix any model M, and fix any minimal witness function π for M. 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 $d = 4 \cdot |\varphi_0|$ (because of Lemma 15(ii)). 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 of node expressions:

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

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

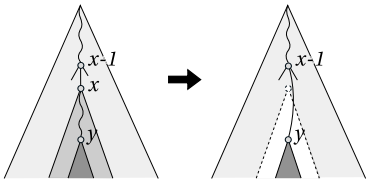
Claim 1: On each branch of M, at most $2^{O(|\varphi_0|^3)}$ distinct types can be realized.

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 the model. Likewise, while $|\mathsf{aux}_{\downarrow^*}(\varphi_0)|$ is $2^{O(|\varphi_0|^2)}$ (cf. Lemma 15), on each single branch in M at most $|\mathsf{aux}_{\downarrow^*}(\varphi_0)|$ many subsets of $\mathsf{aux}_{\downarrow^*}(\varphi_0)$ can be realized. This is due to the volatile natural 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 k-th *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 $2^{O(|\varphi_0|^3)}$.

Finally, consider the node expressions assigned to the nodes by the witness function π. It is not hard to see that, since π is minimal, each $\psi \in \pi(n)$ can be traced back in a finite number of steps to some $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$ being true at an ancestor n' of n. Now, fix a branch in M and let the length of this path be k. Then each of the k nodes can contain at most $|\varphi_0|$ many expressions of the form $\langle\alpha\rangle \in \mathsf{sub}(\varphi_0)$, and each of those expressions will cause at most $O(|\varphi_0|)$ many $\mathsf{aux}(\varphi_0)$-expressions to belong to π-sets of nodes on the branch. Thus, all-together, at most $k \cdot O(|\varphi_0|^2)$ many expressions can occur as elements of π-values of nodes on the branch. This means that, for nodes n on the branch, the average $|\pi(n)|$ will be 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 basic combinatorial fact being that there is no family of j^{i+1} distinct subsets of $\{1,2,\ldots,j\}$ of average cardinality less than i).

Finally, multiplying the numbers, we obtain that at most $2^{O(|\varphi_0|^3)}$ many types can be realized on any branch. *End of proof of claim.*

Claim 2: If M satisfies φ_0 at the root and contains distinct nodes $xR_{\downarrow^*}y$ that have the same type, then the following contraction will preserve truth of φ_0 at the root:



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

Proof of claim: We will prove something stronger, namely that in the contracted model, all remaining nodes still satisfy the same $\mathsf{sub}(\varphi_0)$-expressions ψ as in the original model. The proof goes by induction. The only difficult case is where ψ is of the form $\langle\alpha\rangle$. Let n be any node in the contracted model.

[$\Rightarrow$] Suppose n satisfies $\langle\alpha\rangle$ in the original model. Then there must be a $\beta = (\beta_1/\cdots/\beta_k) \in \mathsf{inst}(\alpha)$ such that $\langle\beta\rangle \in \pi(n)$. One can show by induction on k that n still satisfies $\langle\beta\rangle$ (and hence $\langle\alpha\rangle$) in the contracted model. The base case, where $k=1$, clearly holds. As for the inductive step, we can 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 model, and $\langle\beta_2/\cdots/\beta_k\rangle \in \pi(n)$. It follows by the induction hypothesis, that n satisfies ψ and $\langle\beta_2/\cdots/\beta_k\rangle$, and hence also $\langle\beta\rangle$, in the contracted model.
- $\beta_1 = \downarrow$. 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 model, and hence n satisfies $\langle\beta\rangle$ in the contracted model. 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 model. It follows by the induction hypothesis that y satisfies $\langle\beta_2/\cdots/\beta_k\rangle$ in the contracted model, and therefore n satisfies $\langle\beta\rangle$ in that model.
- $\beta_1 = \downarrow^*$. 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 model, then we can infer from the induction hypothesis that n' satisfies $\langle\beta_2/\cdots/\beta_k\rangle$ in the contracted model, and hence n satisfies $\langle\beta\rangle$ in the contracted model. 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 submodel rooted at y is not affected by the contraction, y must still satisfy $\langle\beta_2/\cdots/\beta_k\rangle$ in the contracted model. We also know that n'' is a descendant of n, and hence n satisfies $\langle\beta\rangle$ in the contracted model.

[$\Leftarrow$] Suppose n satisfies $\langle\alpha\rangle$ in the contracted model. Then there must be a $\beta \in \mathsf{inst}(\alpha)$ such that n satisfies $\langle\beta\rangle$ in the contracted model. We will show that n satisfies $\langle\beta\rangle$ in the original model, 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 model. But then n' is also a descendant of n in the original model, and by induction hypothesis, it satisfies $\langle\vec{\gamma}\rangle$. Therefore, n satisfies $\langle\beta\rangle$ in the original model.

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 $n_0, \ldots, n_k$ be nodes in the contracted model such that $n_0 = n$ and (n_i, n_{i+1}) belongs to β_i in the contracted model, for $i < k$. We distinguish two cases:

- Either $n_i \neq y$ for all $i \leq k$, or $n_0 = y$. In this case, the same path $n_0, \ldots, n_k$ exists also in the original model. Moreover, whenever β_i is of the form $.[\psi]$, then, by induction hypothesis, n_i satisfies ψ in the original model as well. Thus, $\langle\beta\rangle$ must be true at n in the original model.
- $n_i = y$ for some $i \leq k$, and $n_0 \neq y$. It follows that the sequence $n_0, \ldots, n_k$ is actually 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 $n'_0, \ldots, n'_k$ is of the form $y-\ell, \ldots, y-1, y, \ldots, y+m$. Since x and y have the same type in the original model, n_i and n'_i agree on all node expressions from $\mathsf{sub}(\varphi_0)$ (in the original model). Moreover, whenever β_i is of the form $.[\psi]$, then, by induction hypothesis, we know that n_i satisfies ψ in the original model as well. Together this implies that $(y-\ell, y+m)$ satisfies $(\beta_1/\cdots/\beta_k)$ in the original model. Finally, by induction hypothesis, $n'_k = n_k$ satisfies $\langle\vec{\gamma}\rangle$ in the original model. We conclude that n'_0 satisfies $\langle\beta\rangle$ and hence $\langle\alpha\rangle$ in the original model. Finally, using once more the fact that x and y have the same type in the original model, we conclude that n_0 must also satisfy $\langle\alpha\rangle$ in the original model.

End of proof of claim.

Finally, the main argument to establish Lemma 16: let φ_0 be any satisfiable $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression, M a tree model satisfying φ at the root. If M contains a branch of length greater than $2^{p(|\varphi_0|)}$, with p the precise polynomial from Claim 1, then there is a tree with fewer nodes than M in which φ_0 is still satisfied. Since tree models are finite, this shows that φ is satisfiable in a tree model of depth $2^{O(|\varphi_0|^3)}$. □

The following theorem states correctness of the algorithm developed in Section 5.

THEOREM 31. *The algorithm has an accepting run on input φ_0 iff φ_0 is satisfiable.*

PROOF. [⇒] Suppose the algorithm has an accepting run on input φ_0. Let $T = (N, R_{\downarrow}, R_{\rightarrow}, \tau)$ be the recursion tree of one such accepting run, i.e., $(N, R_{\downarrow}, R_{\rightarrow})$ is a finite sibling ordered tree (the sibling order is in fact irrelevant), and τ assigns to each node the complete φ_0-type that is the first argument of the corresponding recursive call of `check`. Take the tree model $M = (N, R_{\downarrow}, R_{\rightarrow}, Lab)$, where $Lab : N \rightarrow \Sigma$ assigns to each node $n \in N$ the unique label from Σ that belongs to $\tau(n)$, if there is one, or otherwise any label not in $\mathsf{sub}(\varphi_0)$.

It can be proved that for all $n \in N$ and $\psi \in \mathsf{cl}(\varphi_0)$, $n \in [\![\psi]\!]^M_{\mathsf{NExpr}}$ iff $\psi \in \tau(n)$. In particular, since $\varphi_0 \in \tau(r)$, for r the root of M, we have that $r \in [\![\varphi_0]\!]^M_{\mathsf{NExpr}}$. 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 the details to the reader.

[⇐] Suppose φ_0 is satisfiable. Then, by Lemma 16, φ_0 is satisfied at the root of a tree model of depth at most $2^{p(|\varphi|)}$. For any node n of this tree model, let $\tau(n) = \{\varphi \in \mathsf{cl}(\varphi_0) \mid n \in [\![\varphi]\!]^M_{\mathsf{NExpr}}\}$. To show that the algorithm has an accepting run on input φ_0, we associate to each call of `check(`t`, `i`)` a "witnessing" node n in the tree, at distance i from the root, satisfying $\tau(n) = t$. Initially, we let the algorithm pick $\tau(r)$, for r the root of the tree model (thus, r itself is the witnessing node). Next, suppose that at some point `check(`t`,`i`)` is called, and let n be a witnessing node (i.e., $\tau(n) = t$ and n lies at distance i from the root), and consider any demand $\varphi \in t$. There must be at least one child n' of n satisfying the remainder φ' of φ (this follows simply from the semantics of CoreXPath). We let the algorithm choose precisely the type $t' = \tau(n')$. In this way, we have that $\varphi' \in t'$, $t \Rightarrow t'$, and n' is a witness for check($t', i+1$). It is easy to see that the algorithm returns 'Yes' on input φ_0 when all non-deterministic choices are made according to this strategy. □

C. MULTI-LABELS

LEMMA 19. *Satisfiability of $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expressions on tree-models with multi-labels can be reduced in polynomial time to satisfiability of $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expressions on standard tree models. The same holds for $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$, $\mathsf{CoreXPath}_{\downarrow\uparrow}(\cap)$ and $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$.*

PROOF. The main idea is as follows: we can turn a multi-labelled tree model into a standard tree model by adding $|Lab(n)|$ extra children to each node n and labelling each with a different element of $Lab(n)$. To distinguish these auxiliary nodes from "real" document nodes, we label the latter with a special node label x.

Following this idea, we can transform every $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression φ into another $\mathsf{CoreXPath}_{\downarrow}(\cap)$ node expression φ', such that φ' is satisfiable in a standard tree model iff φ is satisfiable in a tree model with multi-labels. φ' is defined as follows: let φ^* be obtained from φ by (i) replacing every occurrence of $\downarrow$ or $\downarrow^*$, by $\downarrow[x]$ or $\downarrow^*[x]$, respectively; and (ii) replacing every occurrence of a node label p with $\langle\downarrow[p]\rangle$. Then set $\varphi' := \varphi^* \wedge x \wedge \neg\langle\wedge\neg\langle\downarrow^*[\neg x]/\downarrow\rangle$ (the last conjunct of φ' ensures that all auxiliary nodes are leafs).

Similar reductions can be given for $\mathsf{CoreXPath}_{\downarrow}(*, \cap)$, $\mathsf{CoreXPath}_{\downarrow\uparrow}(\cap)$ and $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$. In the case of $\mathsf{CoreXPath}_{\downarrow\rightarrow}(\cap)$, φ' needs to be extended with an additional conjunct $\neg\langle\downarrow^*[\neg x]/\rightarrow^*[x]\rangle$, ensuring that the auxiliary nodes are always to the right of "real" sibling nodes. □

D. ALTERNATING TURING MACHINES

We introduce alternating Turing machines as used in Section 6. An *Alternating Turing Machine (ATM)* is of the form $\mathcal{M} = (Q, \Lambda, \Gamma, q_0, \Delta)$. The set of *states* $Q = Q_{\exists} \uplus Q_{\forall} \uplus \{q_a\} \uplus \{q_r\}$ consists of *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 Γ the *work alphabet* containing a *blank symbol* $\square$ and satisfying $\Lambda \subseteq \Gamma$; $q_0 \in Q_{\exists} \cup Q_{\forall}$ 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 will 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 configurations $K_1, K_2, \ldots$ such that $K_1 = q_0 w$ and K_{i+1} is a successor configuration of K_i for all $i \geq 0$. 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, only, and refer to [3] for the full definition. Let $\mathcal{M}$ be such an ATM. A halting configuration is *accepting* iff it is of the form $wq_a w'$. For other configurations $K = wqw'$, the acceptance behaviour depends on q: if $q \in Q_\exists$, then K is accepting iff at least one successor configuration is accepting; if $q \in Q_\forall$, then K is accepting iff all successor configurations are accepting. Finally, the ATM $\mathcal{M}$ with starting state q_0 *accepts* the input w iff the *initial configuration* $q_0 w$ is accepting. We use $L(\mathcal{M})$ to denote the language accepted by $\mathcal{M}$, i.e., $L(\mathcal{M}) = \{w \in \Lambda^* \mid \mathcal{M} \text{ accepts } w\}$.