

The Complexity of Reasoning about Pattern-based XML Schemas

Gjergji Kasneci
Max-Planck-Institut für Informatik
Saarbrücken / Germany
kasneci@mpii.mpg.de

Thomas Schwentick
Lehrstuhl Informatik I
Universität Dortmund / Germany
thomas.schwentick@udo.edu

ABSTRACT

In a recent paper, Martens et al. introduced a specification mechanism for XML tree languages, based on rules of the form $r \rightarrow s$, where r, s are regular expressions. Sets of such rules can be interpreted in an existential or a universal fashion. An XML tree is existentially valid with respect to a rule set, if for each node there is a rule such that the root path of the node matches r and the children sequence of the node matches s . It is universally valid if each node matching r also matches s . This paper investigates the complexity of reasoning about such rule sets, in particular the satisfiability and the implication problem. Whereas, in general these reasoning problems are complete for EXPTIME, two important fragments are identified with PSPACE and PTIME complexity, respectively.

Categories and Subject Descriptors

H.2.1 [Database management]: Logical Design

General Terms

Theory

Keywords

XML schemas, Integrity constraints

1. INTRODUCTION

As XML (eXtensible Markup Language) represents a widespread vehicle for data exchange on the Web[8], it is important to exploit the different means for XML schema representation. The most common ones are DTDs (Document Type Definitions) and XSDs (XML Schema Definitions). A DTD is basically a set of rules of the form $a \rightarrow r$ where a is an element name and r is a regular expression over element names. A document is valid with respect to such a rule if each element with name a has a sequence of children that matches r .

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS'07, June 11–13, 2007, Beijing, China.

Copyright 2007 ACM 978-1-59593-685-1/07/0006 ...\$5.00.

One of the disadvantages of DTDs is their limited expressive power. The type of an element depends only on its name. To avoid this shortcoming of DTDs, the W3C (World Wide Web Consortium) proposes the mechanism of XSDs which, besides other extensions, allows a recursive typing of elements. We describe this mechanism in terms of an abstraction, *extended DTDs (EDTDs)*. Basically, an EDTD is a set of rules of the form $\tau \rightarrow r_\tau$ where τ is a type and r_τ is a regular expression over types. Furthermore, each type has an associated element name. A document is valid with respect to an EDTD if each element can be labeled by a compatible¹ type such that, for each element of type τ , the sequence of types of the children is valid w.r.t. r_τ .

Nevertheless, even though the expressive power of EDTDs is much more satisfactory, the notion of types entails a couple of complications. From a users point of view it is more intricate to specify syntactic correctness in terms of types and thus a recursive manner. From a validators it is unpleasant that, in general, type assignment is not unique and even in unambiguous cases type assignment might be impossible in a streaming setting.

In an attempt to strengthen expressive power of schemas while keeping processing efficient, XSDs are EDTDs obeying two restrictions concerning the regular expressions on the right-hand side of rules [8]. The first one is called *Element Declaration Constraint* (EDC) and basically forbids the appearance of different types of the same element in the same regular expression. The second constraint concerns the regular expressions on the right-hand side of the rules. It is called *Unique Particle Attribution* (UPA) and requires a form of determinism for the regular expressions.

Previous work has investigated these constraints from a formal language point of view in terms of EDTDs. In [13] two restrictions of EDTDs were proposed², *single-type* EDTDs and *restrained competition* EDTDs. In a nutshell, single-type EDTDs disallow the occurrence of two types a^1 and a^2 with the same base element a in a regular expression. Restrained competition EDTDs disallow words ua^1v and ua^2w in the language of an expression. It was argued that single-type EDTDs correspond to XSDs. In [11], these classes were further studied and a number of syntactical and semantical characterizations were given. In particular, it was shown that restrained competition EDTDs constitute the class of schemas which allow the computation of the

¹We assume that there are specific types corresponding to each element.

²Actually, [13] used the similar formalism of tree grammars instead of EDTDs.

unique type of an element when its opening tag is met in stream-based processing.

The starting point for this paper is another result of [11]. It was shown that the expressive power of single-type EDTDs can be obtained by a rule-based approach which avoids the complication of recursive types. To this end, *ancestor-guarded DTDs* were studied which are sets of triples (r, a, s) , where a is an element name and r, s are regular expressions over element names. If (r, a, s) and (r', a, s') are two different rules then the languages of r and r' must be disjoint. The semantics of ancestor-guarded DTDs is easiest explained when documents are viewed as node-labeled unranked, sibling-ordered trees. The *root path* of a node is the path from the root to that node. A tree is valid with respect to an ancestor-guarded DTD if, for each node v there is a rule (r, a, s) such that (1) v has label a , (2) the sequence of labels of the root path of v matches r , and (3) the sequence of labels of the children of v matches s . It was shown in [11] that ancestor-guarded DTDs describe the same class of tree languages as single-type EDTDs. An analogous characterization of restrained competition EDTDs in terms of ancestor-sibling-guarded DTDs was given, where the regular expression r describes the root-to-node path, together with the sequences of left siblings of the nodes on that path.

In this paper, we depart slightly from the framework of [11]. We consider rules of the form (r, s) where r again refers to the root path of a node and s to its children sequence. In general, we do not require that rules must have disjoint r -expressions. We investigate existential and universal semantics of rule sets. For existential semantics, for each node v there must be a rule (r, s) such that the root path of v matches r and the children sequence of v matches s . In universal semantics, whenever the root path of a node matches an r the children must match the corresponding s . Note that existential semantics is closer to DTDs whereas universal semantics is more in the spirit of integrity constraints of relational databases. We also consider a setting in which both semantics coexist.

As mentioned above, XML is an important vehicle for data exchange and integration. In these contexts it is an important task to automatically reason about schema constraints. Therefore, we study the complexity of satisfiability and the implication problem for schema rule sets.

Contributions

- We show in Section 3 that, in general, the satisfiability and the implication problem are complete for EXPTIME, for both kinds of semantics, no matter whether r -expressions are disjoint or not. In the remainder of the paper, we investigate restricted rule sets, motivated by practical considerations.
- In Section 4, we restrict r to expressions corresponding to XPath downward expressions with the child and descendant axes and s to deterministic expressions. The latter is in compliance with the restrictions for XSDs. It turns out that for such rule sets the implication problem is complete for PSPACE.
- By a further restriction of the allowed r -expressions, essentially following the outcome of an empirical study reported in [3], we obtain in Section 5 a class of rules with a PTIME implication problem.

Related work. The most closely related work is on static analysis for XPath, more specifically satisfiability and containment in the presence of schemas. Even though, our results and even proofs are similar as previous ones (most notably, an EXPTIME lower bound similar to ours can be found in [15], a similar EXPTIME upper bound in [12]), to the best of our knowledge, they can not directly be concluded from those, as our setting is considerably different. Similar complexities also arise in investigations of integrity constraints (e.g., [1]) and XML data exchange [2]. They particularly differ from our work in that they involve reasoning about attribute values. Our work strongly builds on [11].

Acknowledgement. We thank Wim Martens for various helpful suggestions.

2. DEFINITIONS

2.1 XML documents and trees

As this paper is concerned with the structure of XML documents and not its textual or attribute contents, we model documents by unranked, sibling-ordered trees the nodes of which carry labels from a finite domain Σ . We call such trees Σ -trees. We denote the label of a node v by $\text{lab}(v)$. The subtree rooted at a node v of a tree t is denoted by t_v .

The rules investigated in this paper express properties of paths from the root to a node and about the content of a node, i.e., the sequence of labels of its children. We therefore use the following notations. Let v be a node in a tree. Let $u_0, \dots, u_m$ be the nodes on the path from the root u_0 to $v = u_m$ and let $v_1, \dots, v_n$ be the children of v , ordered from left to right. Then $\text{ch-str}(v)$ denotes $\text{lab}(v_1) \dots \text{lab}(v_n)$ and $\text{a-str}(v)$ denotes $\text{lab}(u_0) \dots \text{lab}(u_m)$. For a tree t , we write $P_{\text{anc}}(t)$ for the set $\{\text{a-str}(v) \text{ch-str}(v) \mid v \in t\}$.

2.2 Pattern-based XML schemas

At first, we define the syntax of pattern-based schemas. In the following we represent the language that corresponds to a regular expression r by $\mathcal{L}(r)$. Let Σ be a finite alphabet of labels.

DEFINITION 2.1. An *ancestor-based schema* P is a set $\{(r_1, s_1), \dots, (r_m, s_m)\}$ where all r_i, s_i are regular expressions over Σ .

Each pair (r_i, s_i) of an ancestor-based schema represents a schema rule. We consider two semantics for ancestor-based schemas.

DEFINITION 2.2. A tree t is *existentially valid* with respect to an ancestor-based schema P if, for every node v of t , there is a rule $(r, s) \in P$ such that $\text{a-str}(v) \in \mathcal{L}(r)$ and $\text{ch-str}(v) \in \mathcal{L}(s)$. In this case, we write $t \models_{\exists} P$.

DEFINITION 2.3. A tree t is *universally valid* with respect to an ancestor-based schema P if, for every node v of t , and each rule $(r, s) \in P$ it holds that $\text{a-str}(v) \in \mathcal{L}(r)$ implies $\text{ch-str}(v) \in \mathcal{L}(s)$. In this case, we write $t \models_{\forall} P$.

We often call a node **valid** if it fulfills the rules of the schema under consideration in the current semantics.

EXAMPLE 2.4. Let $P_1 = \{(a^*, (a+b)^+)\}$, $P_2 = P_1 \cup \{(a^*b, \epsilon)\}$ and $P_3 = P_2 \cup \{(a+b)^*, b\}$. The tree consisting of root a and three leaves with label b is existentially valid with respect to P_2 and P_3 and universally valid with respect to P_1 and P_2 .

We denote the set of Σ -trees which are existentially valid (resp., universally valid) with respect to P by $\mathcal{T}_\Sigma^\exists(P)$ (resp. $\mathcal{T}_\Sigma^\forall(P)$). We often omit the existential or universal quantifier if from the context it is clear which semantics is used. Likewise, we usually drop Σ . In the following, we refer to ancestor-based schemas often simply as schemas.

Motivated by the analogous restriction for DTDs, we will frequently consider the following restriction for expressions on the right-hand sides of rules.

DEFINITION 2.5 ([5]). *A regular expression r is called **one-unambiguous** or **deterministic** if there are no two words wa_iv and wa_jv' in $\mathcal{L}(\bar{r})$ where $\bar{r}$ denotes the expression obtained from r by uniquely indexing each occurrence of a symbol $\sigma \in \Sigma$ in r .*

EXAMPLE 2.6. *For $r = (a+b)^*a(ab)^*$, the expression $\bar{r}$ is $(a_1+b_1)^*a_2(a_3b_2)^*$. As a_1a_2 and a_2 are in $\mathcal{L}(\bar{r})$, by choosing $w = \epsilon$ it is seen that r is not deterministic.*

2.3 Decision problems

Our interest is focused on the following two decision problems concerning schemas.

Satisfiability	
Given:	Schema P , alphabet Σ .
Question:	Is there a non-empty Σ -tree t such that $t \models P$?

Implication	
Given:	Schemas P_1, P_2 , alphabet Σ .
Question:	Does, for each Σ -tree t , $t \models P_1$ imply $t \models P_2$?

Each of these problems can be studied for existential and for universal semantics. We write $P_1 \models_\Sigma^\exists P_2$ if, for each Σ -tree t , $t \models_\Sigma^\exists P_1$ implies $t \models_\Sigma^\exists P_2$ and $P_1 \models_\Sigma^\forall P_2$ if, for each Σ -tree t , $t \models_\Sigma^\forall P_1$ implies $t \models_\Sigma^\forall P_2$. As Σ will normally be clear from the context we usually omit it.

EXAMPLE 2.7. *Schema P_1 from Example 2.4 can not be existentially satisfied, as only nodes labeled with b can occur as leaves but there is no rule for such nodes. On the other hand, P_3 can not be universally satisfied over $\Sigma = \{a, b\}$, as the first b -node on a path from the root must obey two contradictory rules and, on the other hand, there is no satisfying tree consisting only of a -labeled nodes. By the obvious monotonicity properties of the existential and universal semantics, it holds $P_1 \models_\Sigma^\exists P_2 \models_\Sigma^\exists P_3$ and $P_3 \models_\Sigma^\forall P_2 \models_\Sigma^\forall P_1$, but not vice versa.*

Clearly, for the universal semantics, the implication problem is polynomially equivalent to the implication problem where P' consists of a single rule.

2.4 Some basic observations

For a schema P we denote by $\mathcal{L}_\exists(P)$ the set of strings $\{s \in P_{\text{anc}}(t) \mid t \models_\Sigma^\exists P\}$ and by $\mathcal{L}_\forall(P)$ the set $\{s \in P_{\text{anc}}(t) \mid t \models_\Sigma^\forall P\}$. The following proposition shows that these sets are closely tied with the implication problem for ancestor-based schemas.

PROPOSITION 2.8. *Let P, P' be schemas. Then the following holds.*

(a) $P \models_\Sigma^\exists P'$ if and only if $\mathcal{L}_\exists(P) \subseteq \mathcal{L}_\exists(P')$.

(b) $P \models_\Sigma^\forall P'$ if and only if $\mathcal{L}_\forall(P) \subseteq \mathcal{L}_\forall(P')$.

PROOF. (a) Let us assume first that $P \models_\Sigma^\exists P'$. Let $\alpha\beta \in \mathcal{L}_\exists(P)$. Thus there is a tree t and a node v such that $t \models_\Sigma^\exists P$ and $\text{a-str}(v) = \alpha$ and $\text{ch-str}(v) = \beta$. By our assumption $t \models_\Sigma^\exists P'$ and thus $\alpha\beta \in \mathcal{L}_\exists(P')$. Therefore, $\mathcal{L}_\exists(P) \subseteq \mathcal{L}_\exists(P')$.

Let us now assume $\mathcal{L}_\exists(P) \subseteq \mathcal{L}_\exists(P')$. Let t be a tree such that $t \models_\Sigma^\exists P$ and let v be a node of t . Thus, $\text{a-str}(v)\text{ch-str}(v)$ is in $\mathcal{L}_\exists(P)$ and therefore also in $\mathcal{L}_\exists(P')$. It follows that there is a tree $t' \in \mathcal{T}_\Sigma^\exists(P')$ and a node v' of t' with $\text{a-str}(v')\text{ch-str}(v') = \text{a-str}(v)\text{ch-str}(v)$. As $t' \in \mathcal{T}_\Sigma^\exists(P')$, there is a rule of P' which is satisfied by v' and therefore also by v . We can conclude that $t \models_\Sigma^\exists P'$ as desired.

(b) The proof of the ‘only if’ is literally the same as for (a). The other direction is straightforward. $\square$

It is tempting to use Proposition 2.8 to solve the implication problem. E.g., in the existential case, one would construct automata $\mathcal{A}$ and $\mathcal{A}'$ for $\mathcal{L}_\exists(P)$ and $\mathcal{L}_\exists(P')$ and test whether the language of $\mathcal{A}$ is contained in the language of $\mathcal{A}'$. Unfortunately, computing such automata is not as straightforward as one might expect. First, the size of such an automaton can be exponential in the size of the underlying schema. Further, the dependencies between rules can be rather complicated. Nevertheless, in Section 5, we will consider a fragment of patterns for which the implication problem can be efficiently solved along this line.

3. COMPLEXITY OF THE GENERAL IMPLICATION PROBLEM

3.1 Universal semantics

In this section, we establish that the complexity of the satisfiability problem and the implication problem for general pattern-based schemas under universal semantics is complete for EXPTIME. To this end, we show in Proposition 3.1 that the satisfiability problem is hard for EXPTIME and in Proposition 3.3 that the implication problem is in EXPTIME.

Lower bound

The lower bound is by a polynomial-time reduction from the 2-Player Corridor Tiling problem to the satisfiability problem. A **tiling system** is a tuple $D = (T, H, V, \bar{b}, \bar{l}, n)$ where T is a finite set of tiles, $H, V \subseteq T \times T$ are horizontal and vertical constraints, respectively, $\bar{b}, \bar{l}$ are n -tuples of tiles, and n is a natural number. A **corridor tiling** is a mapping $\lambda : \{1, \dots, m\} \times \{1, \dots, n\} \rightarrow T$ for some $m \in \mathbb{N}$ such that $(\lambda(1, 1), \dots, \lambda(1, n)) = \bar{b}$ and $(\lambda(m, 1), \dots, \lambda(m, n)) = \bar{l}$. A tiling is **correct** if it respects the horizontal and the vertical constraints, i.e., for every $i = 1, \dots, m$ and $j = 1, \dots, n - 1$, $(\lambda(i, j), \lambda(i, j + 1)) \in H$, and for every $i = 1, \dots, m - 1$ and $j = 1, \dots, n$, $(\lambda(i, j), \lambda(i + 1, j)) \in V$.

Each tiling system induces a game for two players (I and II) that is played on an $\mathbb{N} \times n$ -board. Each player places tiles in turn. The tiles are placed in successive positions. While player I tries to construct a corridor tiling, player II tries to prevent it. Player I wins if he manages to construct a corridor tiling or when player II makes an illegal move. Thus, there are two decision problems that can be derived from a tiling system.

Corridor Tiling	
Given:	A tiling system D .
Question:	Is there a correct corridor tiling for D ?

2- Player Corridor Tiling	
Given:	Game setting corresponding to a tiling system D .
Question:	Does player I have a winning strategy?

Corridor Tiling is PSPACE-complete, 2-Player Corridor Tiling is EXPTIME-complete [6].

PROPOSITION 3.1. *The satisfiability problem for general pattern-based schemas with universal semantics is EXPTIME-hard.*

PROOF. The proof is by a reduction from 2-Player Corridor Tiling. From a tiling system D , we construct a pattern-based schema P such that there is a tree t with $t \models_{\forall} P$ if and only if player I has a winning strategy in the 2-Player Corridor Tiling game for D . We can assume that player I can enforce that the first row is chosen according to $\bar{b}$ as (1) otherwise II has a winning strategy and (2) this condition can be easily checked.

For two tiles t, t' we write $C(t, t')$ for the set of tiles t'' for which $(t, t'') \in V$ and $(t', t'') \in H$. Further, we write $C(t)$ for the set of tiles t'' with $(t, t'') \in V$. We assume w.l.o.g. that both players have always at least one possible move, i.e., $C(t, t') \neq \emptyset$, for each t, t' .

We first describe how a strategy of player I can be represented by a tree. The tree has three kinds of nodes, I-nodes, II-nodes and $\#$ -nodes. The latter occur at depths which are a multiple of $n + 1$. Between such $\#$ -levels, the levels alternate I-nodes and II-nodes. A I-node represents a move of player I. As the tree corresponds to a strategy of player I, such a node is the unique child of its parent. Correspondingly, below each I-node, there is one II-node, for each possible tile that II can choose. More precisely, each I-node with label t' whose ancestor n levels above has label t has one child labeled t'' , for each $t'' \in C(t, t')$. A strategy tree represents a winning strategy for player I if each path in the tree corresponds to a correct tiling. Thus, the schema P checks that (1) the tree is a strategy tree and (2) it corresponds to a winning strategy for I.

We now describe the schema more formally. Let $D = (T, H, V, \bar{b}, \bar{t}, n)$ be a tiling system, where n is an even number. Let $T = \{\tau_1, \dots, \tau_k\}$. Let $\Sigma = T \cup \{\#\}$, $\bar{b} = (b_1, \dots, b_n)$, $\bar{t} = (t_1, \dots, t_n)$. If $S = \{\sigma_1, \dots, \sigma_m\}$ is a set of tiles, we write S_+ for the expression $\sigma_1 + \dots + \sigma_m$ and $S_{\circ}$ for the expression $\sigma_1 \dots \sigma_m$. We assume for simplicity that the symbols are ordered with respect to some underlying order of the alphabet. We also use superscripts to denote repetition. Thus, e.g., T_+^n denotes the concatenation of n expressions of the form $(\tau_1 + \dots + \tau_k)$. We first give the rules which check the strategy tree property.

- (1) for all $i \leq n$: $(b_1 \dots b_{i-1}, b_i)$
The first row is chosen according to $\bar{b}$
- (2) $((T_+ \#)^* T_+^n, \#)$
At depths which are multiples of $n + 1$ there is a $\#$
- (3) for all $t \in T$: $((T_+ \#)^* t \Sigma_+^n, C(t)_+)$
The moves of player I in the first column are correct

- (4) for all odd i , $1 < i < n$, all $t, t' \in T$:
 $((T_+ \#)^* T_+^{i-1} t \Sigma_+^{n-1} t', C(t, t')_+)$
All other moves of player I are correct
- (5) for all even $i \leq n$, all $t, t' \in T$:
 $((T_+ \#)^* T_+^{i-1} t \Sigma_+^{n-1} t', C(t, t')_{\circ})$
All correct moves of player II are possible

It is easy to see that the rules match their intended meaning. We further note that the languages on the left-hand sides of the rules are pairwise disjoint.

So far there can be no finite tree which satisfies all rules, as no right-hand side language contains the empty string. To this end, we modify rule (3). After a row with label sequence $\bar{t}$, player I can end a path. If one tile is not correct w.r.t. $\bar{t}$, player I has to move otherwise. Thus we replace rule (3) by the following rules.

- (3a) $(T_+ \#)^* t_1 \dots t_n, \epsilon$ After a $\bar{t}$ -row, player I can stop
- (3b) For each $t \neq t_1$: $((T_+ \#)^* t \Sigma_+^n, C(t)_+)$
Player I moves correctly in the first column after a row which did not terminate because of its first symbol
- (3c) For each i , $1 < i \leq n$ $t' \neq t_i$:
 $((T_+ \#)^* t_1 T_+^{i-2} t' T_+^{n-i} \#, C(t_1)_+)$
Player I moves correctly in the first column after a row which did not terminate because of a non-first symbol

It is not hard to show that a tree t is universally valid if and only if it is a winning strategy tree for player I. $\square$

Upper bound

Proposition 3.1 immediately implies an EXPTIME lower bound for the implication problem under universal semantics (via $P \models_{\forall} P'$, for some fixed unsatisfiable schema P'). We show next that the problem can be solved in EXPTIME, thus also giving an EXPTIME upper bound for the satisfiability problem. We will make use of the following definition.

DEFINITION 3.2. A **nondeterministic top-down tree automaton** (NTTA) is a tuple $B = (Q, \Sigma, \delta, q_0)$ where Q is a finite set of states, Σ is an alphabet of labels, q_0 is a starting state, and δ is a function $Q \times \Sigma \rightarrow 2^{Q^*}$ such that for every $q \in Q$ and every $a \in \Sigma$, $\delta(q, a)$ is a regular language over Q .

A **run** of B on a tree t is a mapping $\lambda : \text{Dom}(t) \rightarrow Q$ such that for every node v of t with $v_1, \dots, v_n$ as children nodes holds: $\lambda(v_1) \dots \lambda(v_n) \in \delta(\lambda(v), \text{lab}(v))$. A run of B on t is called **accepting** if for every leaf u of t , $\epsilon \in \delta(\lambda(u), \text{lab}(u))$ holds. A tree t is **accepted** by B if there is an accepting run of B on t . By $\mathcal{L}(B)$ we denote the set of trees accepted by B .

For a gentle introduction to tree automata we refer to [4, 14, 7, 9, 16].

PROPOSITION 3.3. *The implication problem for general pattern-based schemas with universal semantics can be solved in EXPTIME.*

PROOF. We first observe that, for the current universal case, we can test $P \models_{\forall} \{(r, s)\}$ for each rule (r, s) in P' separately, thus we can assume w.l.o.g. that P' has only one rule (r, s) . Given schemas $P = \{(r_1, s_1), \dots, (r_m, s_m)\}$ and

$P' = \{(r, s)\}$, we will construct a NTTA B_P of at most exponential size which accepts all the trees that satisfy P and an NTTA $C_{P'}$ of at most exponential size which accepts all trees that do *not* satisfy P' . By checking whether the intersection of these automata is non-empty (in exponential time in $|P| + |P'|$) we get the desired answer.

In order to obtain B_P we first construct an NTTA B_i for each rule $(r_i, s_i) \in P$. Then B_P is simply the product automaton of all automata B_i , which accepts the intersection of all languages $\mathcal{L}(B_i)$. Let, for each i , $A_i = (Q_{A_i}, \Sigma, \delta_{A_i}, q_{0i}, F_{A_i})$ be a deterministic finite automaton for $\mathcal{L}(r_i)$, and $B_i = (Q_{B_i}, \Sigma, \delta_{B_i}, q_{0i})$ an NTTA with $Q_{B_i} = Q_{A_i} \cup (Q_{A_i} \times \Sigma)$ and δ_{B_i} defined as follows:

- $\delta_{B_i}(q, \sigma) = \mathcal{L}((\delta_{A_i}(q, \sigma))^*)$ for all σ with $\delta_{A_i}(q, \sigma) \notin F$.
- $\delta_{B_i}(q, \sigma) = \{(\delta_{A_i}(q, \sigma), \sigma_1) \dots (\delta_{A_i}(q, \sigma), \sigma_n) \mid \sigma_1 \dots \sigma_n \in \mathcal{L}(s)\}$ if $\delta_{A_i}(q, \sigma) \in F$. In particular: $\epsilon \in \delta_{B_i}(q, a)$ if $\epsilon \in \mathcal{L}(s)$.
- $\delta_{B_i}((q, \sigma), \sigma) = \mathcal{L}((\delta_{A_i}(q, \sigma))^*)$ for all σ with $\delta_{A_i}(q, \sigma) \notin F$.
- $\delta_{B_i}((q, \sigma), \sigma) = \{(\delta_{A_i}(q, \sigma), \sigma_1) \dots (\delta_{A_i}(q, \sigma), \sigma_n) \mid \sigma \in \Sigma \text{ and } \sigma_1 \dots \sigma_n \in \mathcal{L}(s)\}$ if $\delta_{A_i}(q, \sigma) \in F$. In particular: $\epsilon \in \delta_{B_i}((q, \sigma), \sigma)$ if $\epsilon \in \mathcal{L}(s)$.

The size of B_i is at most exponential in the sum of the lengths of r_i and s_i . B_i accepts a tree if and only if it satisfies the rule (r_i, s_i) . Thus, the size of product automaton B_P is “exponential to the m ” and thus also exponential. For the construction of an NTTA $C_{P'}$ which accepts all trees that do not satisfy $P' = \{(r, s)\}$ we make use of the following idea.

The NTTA $C_{P'}$ will accept a tree only if it manages to guess a node v for which $\text{a-str}(v) \in \mathcal{L}(r)$ and $\text{ch-str}(v) \notin \mathcal{L}(s)$. Intuitively, this automaton guesses a path in the tree which leads to such a node v . It enters an accepting state in a leaf v' if (1) the node v was found as an ancestor of v' or (2) the path of v' was not the guessed one. The details are left to the reader. Clearly, the size of $C_{P'}$ is at most exponential in the size of P' . $\square$

We note that Proposition 3.3 can also be shown by translating the rules into Propositional Dynamic Logic in the spirit of [12].

From Propositions 3.1 and 3.3 we immediately get the following theorem.

THEOREM 3.4. *The satisfiability and implication problem for general pattern-based schemas with universal semantics are EXPTIME-complete.*

3.2 Existential semantics and extensions

For existential semantics the same complexity results hold and the proofs are along similar lines. Actually, for the lower bound, the same rules as in the proof of Proposition 3.1 can be used, as for each node in the tree exactly one rule can (and has to) match. The upper bound is via a similar construction of tree automata. It should be noted however that in this case, we cannot assume that P' consists of a singleton rule. Thus, we have the following result.

THEOREM 3.5. *The satisfiability and implication problem for general pattern-based schemas with existential semantics are EXPTIME-complete.*

REMARKS 3.6.

- *The results of this section can be extended to ancestor-sibling based patterns as defined in [11].*
- *The same complexity results also hold for the equivalence problem for pattern-based schemas.*
- *As already mentioned, the lower bounds even hold for schemas in which all left-hand side languages are pairwise disjoint, as witnessed by the construction in Proposition 3.1.*
- *The lower bounds also remain valid if only simple regular expressions are allowed on the left-hand sides of rules: concatenations of factors of the forms (1) $(a_1 + \dots + a_k)$, (2) $(a_1 + \dots + a_k)?$, and (3) $(a_1 + \dots + a_k)^*$.*
- *The lower bounds also remain valid if the regular expressions on the right-hand side are deterministic. This follows easily from the construction in Proposition 3.1.*
- *The lower bounds remain valid if all of the above restrictions are applied at the same time.*
- *The upper bounds can be also generalized to the case of mixed schemas where two sets $P_\exists$ and $P_\forall$ are given and a document is valid if each node is valid with respect to some rule in $P_\exists$ and to all rules in $P_\forall$.*

4. A CLASS OF SCHEMAS WITH XPATH VERTICAL RULES

In Section 3 we showed that the implication problem for general ancestor-based schemas is decidable and pinpointed the exact complexity as EXPTIME-complete. As schemas are usually small, the high complexity does not rule out the possibility to use the algorithm for practically interesting cases. As remarked at the end of the previous section, the complexity is quite robust: The restriction to deterministic right-hand side regular expressions, simple left-hand side regular expressions and disjoint left-hand sides does not decrease the structural complexity.

Nevertheless, as reported in [3], in most XML Schemas that can be obtained from the Web, the type of a node depends at most on the labels of its parent and grand parent. Thus, it makes sense to consider rule sets in which the vertical expressions are of restricted forms. In this section, we study a first such restriction, still leading to PSPACE complexity. In the following section, we will consider a further restriction which still covers all cases occurring in [3], but with PTIME complexity. More precisely, in this section, we require that the left-hand sides of rules correspond to a simple XPath fragment, using only the descendant and child axes.

EXAMPLE 4.1. *child :: b/descendant :: c selects all nodes carrying the label c that have a b-labeled child of the root as ancestor. It corresponds to the regular expression $b\Sigma^*c$.*

We call a regular expression **linear** if it is of the form $w_0\Sigma^* \dots w_{k-1}\Sigma^*w_k$, where w_0 and w_k may be ϵ but the other w_i are non-empty. We call a rule (r, s) linear, if r is linear.

First, we give lower bounds for the complexity of the implication problem. The lower bounds even hold with deterministic right-hand side expressions. In the case of universal

semantics, a PSPACE lower bound for satisfiability follows immediately from a result in [10] stating that it is PSPACE-complete to decide whether, for deterministic regular expressions $s_1, \dots, s_n$, $\mathcal{L}(s_1) \cap \dots \cap \mathcal{L}(s_n) \neq \emptyset$. The latter holds if and only if the schema with the rules $(a, s_1), \dots, (a, s_n)$ is satisfiable.

For existential semantics, we give a PSPACE lower bound for the implication problem. It is tempting to assume that this lower bound follows directly by “complementing” the result we just stated. But this is not directly possible, as the complement of a deterministic regular expression is not necessarily describable by a deterministic regular expression of polynomial size. Nevertheless, since for $P = \{(\Sigma^*, s + \epsilon)\}$ and $P' = \{(\Sigma^*, s_1 + \epsilon), \dots, (\Sigma^*, s_m + \epsilon)\}$ with deterministic regular expressions $s, s_1, \dots, s_m$, $P \models_{\exists} P'$ only holds if $\mathcal{L}(s) \subseteq \bigcup_{i=1}^m \mathcal{L}(s_i)$, it suffices to prove the following result.

PROPOSITION 4.2. *It is PSPACE-hard to decide whether $\mathcal{L}(s) \subseteq \bigcup_{i=1}^m \mathcal{L}(s_i)$ holds for deterministic regular expressions $s, s_1, \dots, s_m$.*

PROOF. The proof is by a reduction from Corridor Tiling³. Given a tiling system D we will construct deterministic regular expressions $s_1, \dots, s_m$, such that $\bigcup_{i=1}^m \mathcal{L}(s_i)$ represents all incorrect corridor tilings, with respect to the horizontal and vertical constraints of the system. Another deterministic regular expression s will represent all corridor tilings having at least a correct form. It will hold $\mathcal{L}(s) \subseteq \bigcup_{i=1}^m \mathcal{L}(s_i)$ if and only if there is no correct corridor tiling with respect to D . In the following we will use analogous notation as in the proof of the Proposition 3.1.

Let $D = (T = \{\sigma_1, \dots, \sigma_k\}, H, V, \vec{b} = (b_1, \dots, b_n), \vec{t} = (t_1, \dots, t_n), n)$ be a tiling system. Without loss of generality we assume that there is no constraint of the form (σ, t_i) with $\sigma \neq t_{i-1}$ or (t_i, σ) with $\sigma \neq t_{i+1}$ in H . Further, we assume that the final row can only appear at an even position. We set $\Sigma = T \times \{1, \dots, n\} \times \{0, 1\}$. While the second component of an element $(\sigma, i, j) \in \Sigma$ represents the potential column in which σ can appear, the third component shows whether the row in which σ can appear is odd (1) or even (0).

For each $(\sigma, i, j) \in \Sigma$ we construct two regular expressions $s_{(\sigma, i, j)}$ and $s'_{(\sigma, i, j)}$. Intuitively, $s_{(\sigma, i, j)}$ describes all strings in which a symbol (σ, i, j) occurs wrongly with respect to a vertical constraint and $s'_{(\sigma, i, j)}$ the strings in which a symbol (σ, i, j) occurs wrongly with respect to a horizontal constraint. To this end let r_{ij} be the expression $(\Sigma - \{(\sigma, i, j)\})^*(\sigma, i, j)$ and let

- $R(\sigma, i, j) = \{(\sigma', i, j') \mid j' = (j+1)(\text{mod } 2), (\sigma, \sigma') \in V\}$,
- $F(\sigma, i, j) = \{(\sigma', h, l) \mid (\sigma, \sigma') \notin V \text{ or } h \neq i \text{ or } l \neq (j+1)(\text{mod } 2)\}$, and
- $s_{(\sigma, i, j)} = r_{ij} \Sigma_+^{n-1} (R(\sigma, i, j) + r_{ij} \Sigma_+^{n-1})^* F(\sigma, i, j) + \Sigma_+^*$.

Accordingly, let

- $R'(\sigma, i, j) = \{(\sigma', i+1, j) \mid (\sigma, \sigma') \in H\}$, for all $i < n$,
- $F'(\sigma, i, j) = \{(\sigma', h, l) \mid (\sigma, \sigma') \notin H \text{ or } h \neq i+1 \text{ or } l \neq j\}$, for all $i < n$, and
- for all $i < n$: $s'_{(\sigma, i, j)} = r_{ij} (R'(\sigma, i, j) + r_{ij})^* F'(\sigma, i, j) + \Sigma_+^*$.

³Corridor Tiling is introduced in Section 3.

Finally, we set:

$$s = (b_1, 1, 1) \dots (b_n, n, 1) ((\Sigma - \{(\sigma, i, 0)\})_+^n)^*(t_1, 1, 0) \dots (t_n, n, 0).$$

It is easy to see that there is no correct corridor tiling with respect to D if and only if

$$\mathcal{L}(s) \subseteq \bigcup_{(\sigma, i, j)} (\mathcal{L}(s_{(\sigma, i, j)}) \cup \mathcal{L}(s'_{(\sigma, i, j)})).$$

□

For the upper bound of the problem we will proceed as follows. Let $P = \{(r_1, s_1), \dots, (r_m, s_m)\}$ and $P' = \{(r'_1, s'_1), \dots, (r'_n, s'_n)\}$. We show that if $P \not\models_{\exists} P'$ there is a tree t of polynomial depth in the sum of the sizes of P and P' which is existentially valid with respect to P but not to P' . It is then fairly straightforward to guess such a tree in polynomial space and to verify that it has the desired properties. More precisely, the tree will be guessed and processed in a depth-first and left-to-right fashion. For each level, the algorithm remembers the (sets of) states of the automata for rules s_i up to the current node. The details are left to the reader. The result then follows from Savitch's theorem, $\text{PSPACE} = \text{NPSpace}$. Thus, it is sufficient to prove the following.

LEMMA 4.3. *If there is a tree t that existentially satisfies P but not P' there is also a tree t' of polynomial depth that existentially satisfies P but not P' .*

PROOF. For a linear expression $r = w_0 \Sigma^* \dots w_{k-1} \Sigma^* w_k$ and each string u , let $F_r(u)$ be the lexicographically maximal pair (c, k) for which $u \in \mathcal{L}(w_0 \Sigma^* \dots \Sigma^* w_{c-1} \Sigma^* x)$, where x is the prefix of w_c of length k .

Let t be a tree with $t \models_{\exists} P$ and $t \not\models_{\exists} P'$. To each node v of t we assign the vector $F(v) = (F_{r_1}(\text{a-str}(v)), \dots, F_{r_m}(\text{a-str}(v)), F_{r'_1}(\text{a-str}(v)), \dots, F_{r'_n}(\text{a-str}(v)))$.

Now let us consider how $F(v)$ can evolve along a path of t from the root to a leaf. First, we observe that the c -component in each $F_r(\text{a-str}(v))$ can never decrease. Thus, with respect to c -components, $F(v)$ takes at most $(|r_1| + \dots + |r_m| + |r'_1| + \dots + |r'_n|)$ many different values along a path. As long as no c -component changes its value, the k -component of $F_{r_i}(\text{a-str}(v))$ only depends on the length of the longest suffix of $\text{a-str}(v)$ that is a prefix of w_j (assuming j is the c -component of $F_{r_i}(\text{a-str}(v))$). It is easy to see that there are at most $(n+m)l$ many different values of the c -components of $F(v)$ along a path, where l denotes the maximum length of a string in one of the r_i or r'_i . Thus, if a path has more than $\omega = (|r_1| + \dots + |r_m| + |r'_1| + \dots + |r'_n|)(n+m)l$ nodes then there must be two such nodes v, v' with $F(v) = F(v')$. Note that in this case, replacing the subtree of t rooted at v by the subtree rooted at v' does not change the validity of any node in the resulting tree with respect to P and P' .

Now let us assume that t has a path of length 3ω . There must be two nodes v and v' on this path for which $F(v) = F(v')$ holds. W.l.o.g. we assume that the distance between v and v' is at most ω and that v lies above v' in t . Let v_w be a witness for the fact that t satisfies P but not P' .

1. If v_w is not a descendant of v or if v_w is a descendant of v' replacing the subtree of t rooted at v by the subtree rooted at v' results in a tree which still verifies that $P \models_{\exists} P'$.
2. If v_w is a descendant of v but not a descendant of v' the length of the path from the root of t to v or from v'

to some leaf is at least ω . Thus, we can again replace a sub-tree by a smaller sub-tree without destroying the witness property of the resulting tree.

As each witness tree of depth $> 3\omega$ can be transformed into a smaller witness tree we can conclude that there must be a witness tree of maximum depth $\leq 3\omega$, thus concluding the lemma. $\square$

By a similar reasoning for the case of universal semantics we can conclude the following.

THEOREM 4.4. *The implication problem for linear schemas under existential and universal semantics is PSPACE-complete.*

Whether the lower bounds can be generalized to the case of disjoint left-hand sides and the precise complexity of satisfiability for existential semantics remain open.

5. A TRACTABLE CLASS OF SCHEMAS

With the goal of getting tractable complexity and with the already mentioned results of [3] in mind, we now study rules which are even more restricted. To this end, we call a regular expression **strongly linear** if it is, for some $k > 0$, of the form $a_1 \cdots a_k$ or $\Sigma^* a_1 \cdots a_k$, where each $a_i \in \Sigma$. Such an expression corresponds to an XPath-expression (in abbreviated notation) $/a_1/\cdots/a_k$ or $//a_1/\cdots/a_k$. We call an ancestor-based schema **strongly linear** if

- (1) it only contains rules (r, s) where r is strongly linear and s is deterministic, and
- (2) whenever (r, s) and (r', s') are two different rules then $\mathcal{L}(r)$ and $\mathcal{L}(r')$ are disjoint.

In this section, we show that for strongly linear schemas the implication problem is tractable for existential semantics (Theorem 5.1) as well as universal semantics (Theorem 5.3).

Before getting to these results, we first convince ourselves that the restriction to pairwise disjoint vertical expressions is indeed necessary to get a tractable fragment (as far as it concerns complexity theory...). Indeed, if we allow that one node fulfills several vertical expressions then the implication problem is at least as hard as testing whether,

- for deterministic regular expressions $s_1, \dots, s_n$, $\mathcal{L}(s_1) \cap \cdots \cap \mathcal{L}(s_n) \neq \emptyset$ (universal semantics), and
- $\mathcal{L}(s) \subseteq \bigcup_{i=1}^m \mathcal{L}(s_i)$ holds for deterministic regular expressions $s, s_1, \dots, s_m$ (existential semantics).

As both problems are PSPACE-complete (cf. Section 4) we get an immediate PSPACE lower bound for all such classes.

Now we turn to the statements and proofs of the two main results of this section. The proofs are similar but not completely analogous. In both cases, the idea is to compute from a schema P a deterministic string automaton $\mathcal{A}_P$ for $\mathcal{L}_\exists(P)$ ($\mathcal{L}_\forall(P)$, resp.). Thus, the implication problem for rule sets can be reduced to the containment problem for deterministic string automata, which is solvable in polynomial time. Recall that the construction of such automata has to take the tree structure into account and, as far we can see, cannot be done by combining the automata for the given regular expressions in a straightforward manner.

THEOREM 5.1. *For strongly linear schemas P, P' it can be decided in polynomial time whether $P \models_\exists P'$.*

This theorem follows directly from Proposition 2.8 and the following proposition.

PROPOSITION 5.2. *There is a polynomial time algorithm which computes, for each strongly linear schema P , a deterministic automaton $\mathcal{A}_P$ for $\mathcal{L}_\exists(P)$.*

PROOF. Before we describe the algorithm, we first illustrate its underlying idea by means of an example. Consider the schema $P_0 = \{(\Sigma^* a, ab + c), (\Sigma^* b, b), (\Sigma^* c, \epsilon)\}$. Clearly, b can not occur in a valid *finite* tree, as each b -node requires another b -node as a child. Consequently, a can not occur as a child of an a -node, as the companion b is not allowed. Thus, $a\$c$ and $ac\$$ are the only strings of $\mathcal{L}_\exists(P_0)$.

Let now $P = \{(r_1, s_1), \dots, (r_n, s_n)\}$ be a strongly linear schema. We assume without loss of generality that there is $m \leq n$ such that $r_i = \Sigma^* w_i$, for $i \leq m$ and $r_i = w_i$, for $i > m$, for suitable strings $w_1, \dots, w_n \in \Sigma^*$. We first describe the construction of $\mathcal{A}_P$ in the case $m = n$, i.e., all r_i have a leading Σ^* term. Let U denote the set of substrings of strings w_i . The algorithm proceeds in two phases.

First, it computes the set S of strings $u \in U$ for which a tree t with a node v exists such that

- $\text{a-str}(v) = u$ and
- each node in t_v is valid.

We illustrate this notion in Figure 1.

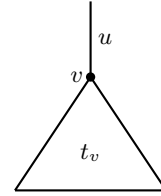


Figure 1: An illustration of the definition of S : all nodes below v must be valid.

To exemplify the algorithm we use the strings $w_1 = aabb$, $w_2 = ab$, $w_3 = aa$ and $w_4 = a$ and the regular expressions $s_1 = \epsilon$, $s_2 = b + c$, $s_3 = b(b + c)^*b$ and $s_4 = a + c$. Here, $S = \{aabb, aab, aa, a\}$.

In the second phase, the algorithm uses S to compute the actual automaton for $\mathcal{L}_\exists(P)$.

We next describe the first phase in more detail. Initially, S contains all strings w_i , for which $\epsilon \in \mathcal{L}(s_i)$. Then the following is repeated until no further changes to S occur:

- for each $u \in U - S$ the set B of symbols σ for which some suffix of $u\sigma$ is (already) in S is computed.
- The string u is added to S if, for some i ,
 - (1) w_i is a suffix of u , and
 - (2) $B^* \cap \mathcal{L}(s_i) \neq \emptyset$.

By induction on the number of iterations of the algorithm, it is straightforward to show that, for each $u \in S$, there are t and v as stated. Conversely, by induction on the depth of the sub-tree rooted at v it can be shown that S eventually contains all such u .

The automaton $\mathcal{A}_P$ is defined as follows. First, let, for each $u \in S$, $i(u)$ denote the (unique) i such that w_i is a

suffix of u and $B(u)$ denote the set of symbols σ for which $u\sigma$ has a suffix in S . Let $\Sigma(u) \subseteq B(u)$ be the set of symbols that occur in some word of $\mathcal{L}(s_{i(u)}) \cap \mathcal{L}(B(u)^*)$. Thus, $\Sigma(u)$ contains exactly those symbols σ , for which there is a tree t and a node v such that

- $\text{a-str}(v) = u$,
- each node in t_v is valid, and
- v has a child labelled with σ .

Note that, if s_i is given by a regular expression, then $B(u)$ and $\Sigma(u)$ actually coincide. The construction presented here also works if $\mathcal{L}(s_i)$ is given by a deterministic automaton.

For each $u \in S$ let $\mathcal{A}_u$ be a deterministic automaton for $\mathcal{L}(s_{i(u)}) \cap \Sigma(u)^*$. Clearly, $\mathcal{A}_u$ can be chosen basically as a sub-automaton of the automaton for s_i and is thus of polynomial size. It might be necessary to add a new rejecting state to which all transitions for non- $\Sigma(u)$ -symbols are directed. The set of states of $\mathcal{A}_P$ consists of $S \cup \{q_0, q_-\}$ and the disjoint union of the states of the automata $\mathcal{A}_u$, where q_0 is a new initial state and q_- is a new sink state. The transition function δ is defined as follows.

- For each symbol σ , $\delta(q_0, \sigma)$ is σ if $\sigma \in S$, otherwise q_- .
- For each $u \in S$ and each symbol $\sigma \in \Sigma(u)$, $\delta(u, \sigma)$ is the longest suffix u' of $u\sigma$ which is in S , and q_- if $\sigma \notin \Sigma(u)$.
- For each $u \in S$, $\delta(u, \$)$ is the initial state of $\mathcal{A}_u$.
- With respect to the states of each $\mathcal{A}_u$, $\mathcal{A}$ is defined just as $\mathcal{A}_u$.
- Finally, $\delta(q_-, \sigma) = q_-$, for each symbol σ .

The only accepting states are those of the automata $\mathcal{A}_u$.

It remains to show that $\mathcal{L}_\exists(P) = \mathcal{L}(\mathcal{A}_P)$.

" $\mathcal{L}_\exists(P) \subseteq \mathcal{L}(\mathcal{A}_P)$ ": Let $\alpha\$ \beta \in \mathcal{L}_\exists(P)$, where $\alpha \in \Sigma^+$, $\beta \in \Sigma^*$. Thus, there is a tree t with a node v such that $\text{a-str}(v) = \alpha$ and $\text{ch-str}(v) = \beta$. Thus, there is a (unique) i such that w_i is a suffix of α and $\beta \in \mathcal{L}(s_i)$. In particular, from t a witness to $w_i \in S$ can be extracted. Likewise, every prefix of α has a suffix in S . Thus, after reading α , $\mathcal{A}_P$ will be in state w_i , when reading ch-str it will enter the initial state of $\mathcal{A}_i$ and finally it will accept $\alpha\$ \beta$.

" $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}_\exists(P)$ ": Let $\alpha\$ \beta \in \mathcal{L}(\mathcal{A}_P)$, where $\alpha = \alpha_1 \cdots \alpha_k$, $\alpha_i \in \Sigma$. We construct a tree witnessing $\alpha\$ \beta \in \mathcal{L}_\exists(P)$ in a top-down fashion. More precisely, for each i , $1 \leq i \leq k$, we construct a tree t_i with a node v_i such that $\text{a-str}(v_i) = \alpha_i \cdots \alpha_k$, $\text{ch-str}(v_i) = \epsilon$ and all nodes besides v_i are valid. For $i = 1$ this is straightforward⁴: We can let t_1 be the tree with one node, labelled with α_1 .

Thus, let $i \geq 1$ and t_i already be constructed. Let u be the longest suffix of $\alpha_1 \cdots \alpha_i$ in U and j such that w_j is some suffix of u . Thus, $\mathcal{A}_P$ is in state u after reading $\alpha_1 \cdots \alpha_i$. As $\mathcal{A}_P$ accepts $\alpha\$ \beta$, we can conclude that $\alpha_{i+1} \in \Sigma(u)$ and there is a string $x \in \mathcal{L}(s_j) \cap \Sigma(u)^*$ containing α_{i+1} . For each symbol σ of x , $u\sigma$ has a suffix u_σ in U . Therefore, there is a tree t_σ with a node v_σ such that $\text{a-str}(v_\sigma) = u_\sigma$ and every node in the subtree rooted at v_σ is valid. By combining copies of these trees we get a tree t' with a node v such that $\text{a-str}(v) = \alpha_1 \cdots \alpha_i$, $\text{ch-str}(v) = x$ and all nodes in t'_v are

⁴As the string is accepted, $\alpha_1 \in S$.

valid. By gluing together t' and t_i along the root path of v and deleting the subtree below some α_{i+1} -labelled child of v we obtain t_{i+1} .

Finally, from t_k we get t by an analogous process, adding children with label sequence β below v together with respective subtrees. The reasoning is just as in the induction step above.

Thus, we have shown the theorem for $m = n$. If $m < n$, we have to deal with expressions $r_i = w_i$. It is straightforward, though, to generalize the construction of $\mathcal{A}_P$ for this case. Basically, $\mathcal{A}_P$ can have more accepting runs. More precisely, after reading a prefix α of the input, even if the current state u is not in U , $\mathcal{A}_P$ might go on, if $\alpha = w_i$, for some $i > m$. Clearly, if $\$$ is the next symbol it switches to $\mathcal{A}_i$. $\square$

Now we turn to the result for universal semantics.

THEOREM 5.3. *For strongly linear schemas P, P' it can be decided in polynomial time whether $P \models_{\forall} P'$.*

Thanks to Proposition 2.8 it is sufficient to prove the following proposition.

PROPOSITION 5.4. *There is a polynomial time algorithm which computes, for each strongly linear schema P , a deterministic automaton $\mathcal{A}_P$ for $\mathcal{L}_{\forall}(P)$.*

PROOF. The proof is similar to the one of Proposition 5.2 but in some aspects different, due to a fundamental difference between the existential and the universal semantics: consider a witness tree t as displayed in Figure 1. In the case of existential semantics, no matter what is added above the root v of such a tree, it retains the property that all nodes in t_v are valid. For universal semantics, this might not be true as, extending the root path beyond the root might make more left-hand side expressions of rules match nodes in t_v . As an example, consider the schema consisting of $(\Sigma^* a, c), (\Sigma^* bac, b)$. In the two-node tree with an a -labelled root and a node v with label c , v is valid, but adding a node labelled b above the root makes it invalid, as v does not have a child with label b . Therefore, in a pre-processing step, the schema is transformed to an equivalent one, in which this problem can not arise. To this end, let $P = \{(r_1, s_1), \dots, (r_n, s_n)\}$ be a strongly linear schema. Let m and $w_1, \dots, w_n$ be chosen as in the proof of Proposition 5.2. The pre-processing step works as follows. Let $i \neq j$, $r_i = \Sigma^* w_i$ and $r_j = \Sigma^* w_j$, and $w_j = uw_i u'$, with $u \neq \epsilon$. By the disjointness requirement, thus $u' \neq \epsilon$. Let $u = a_1 \cdots a_k$. Then for each j , $1 \leq j \leq k$ and each $b \in \Sigma - \{a_j\}$ the rule $(\Sigma^* ba_j \cdots a_k w_i, s_i)$ is added to P . Furthermore, for each $j \leq k + 1$, the rule $(a_j \cdots a_k w_i, s_i)$ is added. Finally, $(\Sigma^* a_1 \cdots a_k w_i, s_i)$ is added and $(\Sigma^* w_i, s_i)$ is deleted. This process is repeated until no such w_i, w_j exist in P . We refer to the resulting schema again by P .

For the example schema, we get the transformed schema $(\Sigma^* bac, b), (\Sigma^* aa, c), (\Sigma^* ca, c), (a, c)$.

It should be noted, that in the case of $u = \epsilon$, no transformation is needed. It should also be noted, that the size of P increases only polynomially as each new left-hand side of a rule can be obtained by changing (or deleting) one symbol in a sub-string of an old left-hand side. Furthermore, the set of valid trees is not affected by the translation. Thus, we assume now a schema in which no string w_i is a substring of a w_j , $i \neq j$ (unless it is a prefix). As in Proposition 5.2, we first consider the case where $m = n$, i.e., there are no

rules $r_i = w_i$. Let N denote the set of strings which do *not* match any left-hand side expression of P , i.e., they do not have any w_i as a suffix.

Again, the algorithm has three phases. In its first phase, the algorithm computes the set S of all w_i for which there is a tree t with a node v such that

- $\text{a-str}(v) = w_i$ and
- each node in t_v is valid.

Initially, S contains all strings w_i , for which $\epsilon \in \mathcal{L}(s_i)$. Then the following is repeated until no further changes to S occur: for each $w_i \notin S$ the algorithm computes the set $B(i)$ of symbols b occurring in s_i for which $w_i b$ is in N or has a suffix in S . The string w_i is added to S if $\mathcal{L}(s_i) \cap B(i)^* \neq \emptyset$. As in Proposition 5.2, it can be shown by induction that the set S computed by the first phase matches its specification. In the following, we write U for the set of strings that have a suffix in S .

In the second phase, for each w_i , $\Sigma(i)$ is the set of all symbols $\sigma \in B(i)$, for which $\mathcal{L}(s_i) \cap B(i)^*$ contains at least one string with symbol σ .

Finally, the algorithm computes $\mathcal{A}_P$. Let first W be the set of all strings $u \in N \cup U$ which are a prefix of some w_i . Furthermore, for each $w_i \in S$, let $\mathcal{A}_i$ again be a deterministic automaton for $\mathcal{L}(s_i) \cap \Sigma(i)^*$. Furthermore, for each $u \in W$, let $\mathcal{A}_u$ be an automaton for $\Sigma(u)^*$, where $\Sigma(u)$ contains all symbols σ such that $u\sigma \in N \cup U$.

The set of states of $\mathcal{A}_P$ consists of $W \cup \{q_-\}$, where q_- is a new state, and the disjoint union of the states of the automata $\mathcal{A}_i$ and $\mathcal{A}_u$. Note that the number of states is polynomial in $|P|$. The initial state is ϵ . The transition function δ is defined as follows.

- For each state $u \in W$ and symbol $\sigma \in \Sigma$, $\delta(u, \sigma)$ is
 - the longest suffix of $u\sigma$ that is a prefix of some w_i , in case $u\sigma \in N \cup U$,
 - q_- , otherwise.
- For each state $u \in W$, $\delta(u, \$)$ is
 - the initial state of $\mathcal{A}_i$, if u has suffix⁵ w_i ,
 - the initial state of $\mathcal{A}_u$, otherwise⁶.
- For the states of the automata $\mathcal{A}_i$ and $\mathcal{A}_u$, δ is just copied from these automata.
- Finally, $\delta(q_-, \sigma) = q_-$, for each symbol σ .

The only accepting states are the accepting states of the automata $\mathcal{A}_i$ and $\mathcal{A}_u$.

In a similar fashion as in Proposition 5.2, it can be shown that $L(\mathcal{A}_P) = \mathcal{L}_\forall(P)$. Finally, it is again straightforward to adapt the construction for the case $m < n$. $\square$

6. CONCLUSION

The results of this paper show that the simplicity of describing XML languages by ancestor-based schemas comes at a price: the complexity of basic reasoning tasks is EXPTIME-complete as opposed to PSPACE for single-type EDTDs. Nevertheless, for the vast majority of schemas,

⁵Note that in this case, w_i is in S , by construction of δ .

⁶In this case, $u \in U - N$.

where types only depend on the direct ancestors of a node, and expressions for the children are deterministic, the implication problem can be solved in polynomial time. We leave the precise complexity of linear schemas with disjoint left-hand sides as an open problem.

7. REFERENCES

- [1] M. Arenas and L. Libkin. A normal form for XML documents. *ACM Trans. on Database Systems*, 29(1):195–232, 2004.
- [2] M. Arenas and L. Libkin. Xml data exchange: Consistency and query answering. In *PODS 05*, pages 13–24, 2005.
- [3] G. J. Bex, W. Martens, F. Neven, and T. Schwentick. Expressiveness of XSDs: from practice to theory, there and back again. *WWW 05*, pages 712–721, 2005.
- [4] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular Tree and Regular Hedge Languages over Unranked Alphabets. *Technical Report HKUST-TCSC-2001-0*, The Hongkong University of Science and Technology, Version 1, 3. April 2001.
- [5] A. Brüggemann-Klein and D. Wood. One-Unambiguous Regular Languages. *Information and Computation*, 142(2):182–206, 1998.
- [6] B. S. Chlebus. Domino-Tiling Games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986.
- [7] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. <http://www.grappa.univ-lille3.fr/tata>, 1997.
- [8] World Wide Web Consortium. *Extensible Markup Language (XML)*. <http://www.w3.org/XML>.
- [9] J. Cristau, C. Löding, and W. Thomas. Deterministic Automata on Unranked Trees. In *FCT 05*, pages 68–79, 2005.
- [10] W. Martens, F. Neven, and T. Schwentick. Complexity of Decision Problems for Simple Regular Expressions. *MFCS 04*, pages 889–900, 2004.
- [11] W. Martens, F. Neven, and T. Schwentick. Which XML Schemas Admit 1-Pass Preorder Typing? *ICDT 05*, pages 68–82, 2005.
- [12] M. Marx. XPath with conditional axis relations. In *EDBT 04*, 2004.
- [13] M. Murata, D. Lee, and M. Mani. Taxonomy of XML Schema Languages using Formal Language Theory. *Extreme Markup Languages, Montreal, Canada*, 2001.
- [14] F. Neven. Automata, Logic, and XML. *CSL 02*, pages 2–26, 2002.
- [15] 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(3), 2006.
- [16] T. Schwentick. Automata for XML - A Survey. *Journal of Computer and System Sciences*, 73(3):289–315, 2007.