Journal of Computer and System Sciences 77 (2011) 505–519

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Succinctness of pattern-based schema languages for XML ☆

Wouter Gelade *,[1], Frank Neven

Hasselt University and Transnational University of Limburg, School for Information Technology, Belgium

ARTICLE INFO

Article history:
Received 31 January 2008
Received in revised form 16 September 2008
Available online 24 April 2010

Keywords:
Complexity
Schema transformation
XML schema languages

ABSTRACT

Martens et al. defined a pattern-based specification language equivalent in expressive power to the widely adopted XML Schema definitions (XSDs). This language consists of rules of the form (r, s) where r and s are regular expressions and can be seen as a type-free extension of DTDs with vertical regular expressions. Sets of such rules can be interpreted both in an existential or universal way. In the present paper, we study the succinctness of both semantics w.r.t. each other and w.r.t. the common abstraction of XSDs in terms of single-type extended DTDs. The investigation is carried out relative to three kinds of vertical pattern languages: regular, linear, and strongly linear patterns. We also consider the complexity of the simplification problem for each of the considered pattern-based schemas.

1. Introduction

In formal language theoretic terms, an XML schema defines a tree language. The for historical reasons still widespread Document Type Definitions (DTDs) can then be seen as context-free grammars with regular expressions at right-hand sides which define the local tree languages [1]. XML Schema [2] extends the expressiveness of DTDs by a typing mechanism allowing content-models to depend on the type rather than only on the label of the parent. Unrestricted application of such typing leads to the robust class of unranked regular tree languages [1] as embodied in the XML schema language Relax NG [3]. The latter language is commonly abstracted in the literature by extended DTDs (EDTDs) [4]. The Element Declarations Consistent constraint in the XML Schema specification, however, restricts this typing: it forbids the occurrence of different types of the same element in the same content model. Murata et al. [5] therefore abstracted XSDs by single-type EDTDs. Martens et al. [6] subsequently characterized the expressiveness of single-type EDTDs in several syntactic and semantic ways. Among them, they defined an extension of DTDs equivalent in expressiveness to single-type EDTDs: ancestor-guarded DTDs. An advantage of this language is that it makes the expressiveness of XSDs more apparent: the content model of an element can only depend on regular string properties of the string formed by the ancestors of that element. Ancestor-based DTDs can therefore be used as a type-free front-end for XML Schema. As they can be interpreted both in an existential and universal way, we study in this paper the complexity of translating between the two semantics and into the formalisms of DTDs, EDTDs, and single-type EDTDs.

In the remainder of the paper, we use the name pattern-based schema, rather than ancestor-based DTD, as it emphasizes the dependence on a particular pattern language. A pattern-based schema is a set of rules of the form (r, s), where r and s are regular expressions. An XML tree is then existentially valid w.r.t. a rule set if for each node there is a rule such that the path from the root to that node matches r and the child sequence matches s. Furthermore, it is universally valid if each node vertically matching r, horizontally matches s. The existential semantics is exhaustive, fully specifying every

☆ An extended abstract of this paper appeared in the Proceedings of the 11th Biennial Symposium on Data Base Programming Languages (DBPL 2007).

* Corresponding author.
E-mail addresses: wouter.gelade@uhasselt.be (W. Gelade), frank.neven@uhasselt.be (F. Neven).

[1] Research Assistant of the Fund for Scientific Research – Flanders (Belgium).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2010.04.008

Table 1
Overview of complexity results for translating pattern-based schemas into other schema formalisms. For all non-polynomial complexities, except the ones marked with a star, there exist examples matching this upper bound. Theorem numbers are given in parentheses.

	Other semantics	EDTD	EDTDst	DTD
$\mathcal{P}_\exists$(Reg)	2-exp (14(1))	exp (14(2))	exp (14(3))	exp* (14(5))
$\mathcal{P}_\forall$(Reg)	2-exp (14(6))	2-exp (14(7))	2-exp (14(8))	2-exp (14(10))
$\mathcal{P}_\exists$(Lin)	\ (16(1))	exp (16(2))	exp (16(3))	exp* (16(5))
$\mathcal{P}_\forall$(Lin)	\ (16(6))	2-exp (16(7))	2-exp (16(8))	2-exp (16(10))
$\mathcal{P}_\exists$(S-Lin)	poly (20(1))	poly (20(2))	poly (20(3))	poly (20(6))
$\mathcal{P}_\forall$(S-Lin)	poly (20(7))	poly (20(8))	poly (20(9))	poly (20(12))
$\mathcal{P}_\exists$(Det-S-Lin)	poly (20(1))	poly (20(2))	poly (20(3))	poly (20(6))
$\mathcal{P}_\forall$(Det-S-Lin)	poly (20(7))	poly (20(8))	poly (20(9))	poly (20(12))

Table 2
Overview of complexity results for pattern-based schemas. All results, unless indicated otherwise, are completeness results. Theorem numbers for the new results are given in parentheses.

	SIMPLIFICATION	SATISFIABILITY	INCLUSION
$\mathcal{P}_\exists$(Reg)	EXPTIME (14(4))	EXPTIME [7]	EXPTIME [7]
$\mathcal{P}_\forall$(Reg)	EXPTIME (14(9))	EXPTIME [7]	EXPTIME [7]
$\mathcal{P}_\exists$(Lin)	PSPACE (16(4))	PSPACE [7]	PSPACE [7]
$\mathcal{P}_\forall$(Lin)	PSPACE (16(9))	PSPACE [7]	PSPACE [7]
$\mathcal{P}_\exists$(S-Lin)	PSPACE (20(4))	PSPACE [7]	PSPACE [7]
$\mathcal{P}_\forall$(S-Lin)	PSPACE (20(10))	PSPACE [7]	PSPACE [7]
$\mathcal{P}_\exists$(Det-S-Lin)	in PTIME (20(5))	in PTIME [7]	in PTIME [7]
$\mathcal{P}_\forall$(Det-S-Lin)	in PTIME (20(11))	in PTIME [7]	in PTIME [7]

allowed combination, and more DTD-like, whereas the universal semantics is more liberal, enforcing constraints only where necessary.

Kasneci and Schwentick studied the complexity of the satisfiability and inclusion problem for pattern-based schemas under the existential (∃) and universal (∀) semantics [7]. They considered regular (Reg), linear (Lin), and strongly linear (S-Lin) patterns. These correspond to the regular expressions, XPath-expressions with only child (/) and descendant (//), and XPath-expressions of the form $//w$ or $/w$, respectively. Deterministic strongly linear (Det-S-Lin) patterns are strongly linear patterns in which additionally all horizontal expressions s are required to be one-unambiguous deterministic [12]. A snapshot of their results is given in the third and fourth column of Table 2. These results indicate that there is no difference between the existential and universal semantics.

We, however, show that with respect to succinctness there is a huge difference. Our results are summarized in Table 1. Both for the pattern languages Reg and Lin, the universal semantics is exponentially more succinct than the existential one when translating into (single-type) extended DTDs and ordinary DTDs. Furthermore, our results show that the general class of pattern-based schemas is ill-suited to serve as a front-end for XML Schema due to the inherent exponential or double exponential size increase after translation. Only when resorting to S-Lin patterns, there are translations only requiring polynomial size increase. Fortunately, the practical study in [6] shows that the sort of typing used in XSDs occurring in practice can be described by such patterns. Our results further show that the expressive power of the existential and the universal semantics coincide for Reg and S-Lin, albeit a translation cannot avoid a double exponential size increase in general in the former case. For linear patterns the expressiveness is incomparable. Finally, as listed in Table 2, we study the complexity of the simplification problem: given a pattern-based schema, is it equivalent to a DTD?

Outline. The paper is further organized as follows. In Section 2, we recall the necessary definitions concerning regular expressions, schema languages, and pattern-based schemas. We define the decision problems we consider and introduce a notation for succinctness. In Sections 3, 4, and 5, we study pattern-based schemas with regular, linear, and strongly linear expressions, respectively. We conclude in Section 6.

2. Preliminaries

In this section, we recall the necessary definitions and results concerning regular expressions, schema languages for XML and pattern-based schemas. We also formally define the problems we address.

2.1. Regular expressions

For the rest of the paper, Σ always denotes a finite alphabet. A *Σ-symbol* (or simply symbol) is an element of Σ, and a *Σ-string* (or simply string) is a finite sequence $w = a_1 \cdots a_n$ of Σ-symbols. We define the length of w, denoted by $|w|$, to be n. We denote the empty string by ε. The set of *positions of w* is $\{1, \ldots, n\}$ and the *symbol of w at position i* is a_i. By $w_1 \cdot w_2$ we denote the *concatenation* of two strings w_1 and w_2. For readability, we usually denote the concatenation of w_1

and w_2 by $w_1 w_2$. The set of all strings is denoted by Σ^* and the set of all non-empty strings by Σ^+. A *string language* is a subset of Σ^*. For two string languages $L, L' \subseteq \Sigma^*$, we define their concatenation $L \cdot L'$ to be the set $\{w \cdot w' \mid w \in L,\ w' \in L'\}$. We abbreviate $L \cdot L \cdots L$ (i times) by L^i.

The set of *regular expressions* over Σ, denoted by RE, is defined in the usual way: $\emptyset$, ε, and every Σ-symbol is a regular expression; and when r_1 and r_2 are regular expressions, then $r_1 \cdot r_2$, $r_1 + r_2$, and r_1^* are also regular expressions. The language defined by a regular expression r, denoted by $L(r)$, is inductively defined as follows: $L(\emptyset) = \emptyset$; $L(\varepsilon) = \{\varepsilon\}$; $L(a) = \{a\}$; $L(r_1 r_2) = L(r_1) \cdot L(r_2)$; $L(r_1 + r_2) = L(r_1) \cup L(r_2)$; and $L(r^*) = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} L(r)^i$. The *size* of a regular expression r over Σ, denoted by $|r|$, is the number of Σ-symbols and operators occurring in r. By $r?$, r^+, and r^k, with $k \in \mathbb{N}$, we abbreviate the expression $r + \varepsilon$, rr^*, and $rr \cdots r$ (k times), respectively. For a set $S = \{a_1, \ldots, a_n\} \subseteq \Sigma$, we denote by S^* the regular expression $(a_1 + \cdots + a_n)^*$. The sets of prefixes and suffixes of strings defined by r are $\mathrm{Prefix}(r) = \{w \mid \exists v \in \Sigma^*,\ wv \in L(r)\}$ and $\mathrm{Suffix}(r) = \{w \mid \exists v \in \Sigma^*,\ vw \in L(r)\}$.

To indicate different occurrences of the same symbol in a RE, we mark symbols with subscripts. For instance, the *marking* of $(a+b)^*a+bc$ is $(a_1+b_2)^*a_3+b_4c_5$. We denote by $r^\flat$ the marking of r and by $\mathrm{Sym}(r^\flat)$ the subscripted symbols occurring in $r^\flat$. When r is a marked expression, then $r^\natural$ over Σ is obtained from r by dropping all subscripts. This notion is extended to words and languages.

A regular expression r is 1-*unambiguous* iff for all words $w, u, v \in \mathrm{Sym}(r^\flat)^*$, and all symbols $x, y \in \mathrm{Sym}(r^\flat)$, the conditions $uxv, uyw \in L(r^\flat)$ and $x \neq y$ imply $x^\natural \neq y^\natural$.

A non-deterministic finite automaton (NFA) A is a 4-tuple (Q, q_0, δ, F) where Q is the set of states, q_0 is the initial state, F is the set of final states and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. We write $q \Rightarrow_{A,w} q'$ when w takes A from state q to q'.

We use the following theorem of Glaister and Shallit [8].

Theorem 1. *(See [8].) Let $L \subseteq \Sigma^*$ be a regular language and suppose there exists a set of pairs $M = \{(x_i, w_i) \mid 1 \leqslant i \leqslant n\}$ such that*

- *$x_i w_i \in L$ for $1 \leqslant i \leqslant n$; and*
- *$x_i w_j \notin L$ for $1 \leqslant i, j \leqslant n$ and $i \neq j$.*

Then any NFA accepting L has at least n states.

We make use of the following results on transformations of regular expressions. Theorem 2(3)–(4) are from [9].

Theorem 2.

(1) *Let $r_1, \ldots, r_n, s_1, \ldots, s_m$ be regular expressions. A regular expression r, with $L(r) = \bigcap_{i \leqslant n} L(r_i) \setminus \bigcup_{i \leqslant m} L(s_i)$, can be constructed in time double exponential in the sum of the sizes of all r_i, s_j, $i \leqslant n$, $j \leqslant m$.*
(2) *Let $r_1, \ldots, r_n$ be regular expressions. A regular expression r, with $L(r) = \bigcap_{i \leqslant n} L(r_i)$, can be constructed in time double exponential in the sum of the sizes of all r_i, $i \leqslant n$.*
(3) *For every $n \in \mathbb{N}$, there are a linear number of regular expressions $r_1, \ldots, r_m$ of size linear in n such that any regular expression r with $L(r) = \bigcap_{i \leqslant m} L(r_i)$ must be of size at least double exponential in n.*
(4) *For every $n \in \mathbb{N}$, there is a regular expression r_n of size linear in n such that any regular expression r defining $\Sigma^* \setminus L(r_n)$ is of size at least double exponential in r.*
(5) *For any regular expressions r and alphabet $\Delta \subseteq \Sigma$, an expression r^-, such that $L(r^-) = L(r) \cap \Delta^*$, can be constructed in time linear in the size of r.*

Proof. (1) First, for every $i \leqslant n$, construct an NFA A_i, such that $L(r_i) = L(A_i)$. This can be done in polynomial time using for instance the Glushkov construction [10]. Then, let A be the DFA accepting $\bigcap_{i \leqslant n} L(A_i)$ obtained from the A_i by determinization followed by a product construction. For k the size of the largest NFA, this can be done in time $\mathcal{O}(2^{k \cdot n})$. For every $i \leqslant m$, construct an NFA B_i, with $L(s_i) = B_i$, and let B_i be the DFA accepting $\bigcup_{i \leqslant m} L(B_i)$ again obtained from the B_i by means of determinization and a product construction. Similarly, B can also be computed in time exponential in the size of the input. Then, compute the DFA B' for the complement of B by making B complete and exchanging final and non-final states in B, which can be done in time polynomial in the size of B. Then, the DFA C accepts $L(A) \cap L(B')$ and can again be obtained by a product construction on A and B' which requires polynomial time in the sizes of A and B'. Therefore, C is of exponential size in function of the input. Finally, r, with $L(r) = \bigcap_{i \leqslant n} L(r_i) \setminus \bigcup_{i \leqslant m} L(s_i)$, is obtained from C by means of state elimination. This can be done in time exponential in the size of C and thus yields a double exponential algorithm in total.

(2) This follows immediately from Theorem 2(1) by taking $m = 1$ and $s_1 = \emptyset$.

(5) The algorithm proceeds in two steps. First, replace every symbol $a \notin \Delta$ in r by $\emptyset$. Then, use the following rewrite rules on subexpressions of r as often as possible: $\emptyset^* = \varepsilon$, $\emptyset s = s\emptyset = \emptyset$, and $\emptyset + s = s + \emptyset = s$. This gives us r^- which is equal to $\emptyset$ or does not contain $\emptyset$ at all, with $L(r^-) = L(r) \cap \Delta^*$. □

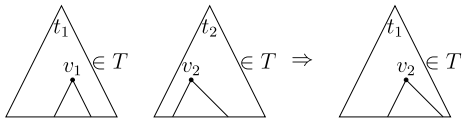

Fig. 1. Closure under label-guarded subtree exchange.

2.2. Schema languages for XML

The set of *unranked Σ-trees*, denoted by $\mathcal{T}_\Sigma$, is the smallest set of strings over Σ and the parenthesis symbols "(" and ")" such that, for $a \in \Sigma$ and $w \in (\mathcal{T}_\Sigma)^*$, $a(w)$ is in $\mathcal{T}_\Sigma$. So, a tree is either ε (empty) or is of the form $a(t_1 \cdots t_n)$ where each t_i is a tree. In the tree $a(t_1 \cdots t_n)$, the subtrees $t_1, \ldots, t_n$ are attached to the root labeled a. We write a rather than $a()$. Notice that there is no a priori bound on the number of children of a node in a Σ-tree; such trees are therefore *unranked*. For every $t \in \mathcal{T}_\Sigma$, the *set of nodes* of t, denoted by $\mathrm{Dom}(t)$, is the set defined as follows: (i) if $t = \varepsilon$, then $\mathrm{Dom}(t) = \emptyset$; and (ii) if $t = a(t_1 \cdots t_n)$, where each $t_i \in \mathcal{T}_\Sigma$, then $\mathrm{Dom}(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{iu \mid u \in \mathrm{Dom}(t_i)\}$. For a node $u \in \mathrm{Dom}(t)$, we denote the label of u by $\mathrm{lab}^t(u)$. By $\text{anc-str}^t(u)$ we denote the sequence of labels on the path from the root to u including both the root and u itself, and $\text{ch-str}^t(u)$ denotes the string formed by the labels of the children of u, i.e., $\mathrm{lab}^t(u1) \cdots \mathrm{lab}^t(un)$. In the sequel, whenever we say tree, we always mean Σ-tree. Denote by $t_1[u \leftarrow t_2]$ the tree obtained from a tree t_1 by replacing the subtree rooted at node u of t_1 by t_2. By $\text{subtree}^t(u)$ we denote the subtree of t rooted at u. A *tree language* is a set of trees.

We make use of the following definitions to abstract from the commonly used schema languages [6]:

Definition 3. Let $\mathcal{R}$ be a class of representations of regular string languages over Σ.

(1) A DTD($\mathcal{R}$) over Σ is a tuple (Σ, d, s_d) where d is a function that maps Σ-symbols to elements of $\mathcal{R}$ and $s_d \in \Sigma$ is the start symbol. For notational convenience, we sometimes denote (Σ, d, s_d) by d and leave the start symbol s_d implicit.
A tree t *satisfies* d if (i) $\mathrm{lab}^t(\varepsilon) = s_d$ and, (ii) for every $u \in \mathrm{Dom}(t)$ with n children, $\mathrm{lab}^t(u1) \cdots \mathrm{lab}^t(un) \in L(d(\mathrm{lab}^t(u)))$. By $L(d)$ we denote the set of trees satisfying d.
(2) An *extended DTD* (EDTD($\mathcal{R}$)) over Σ is a 5-tuple $D = (\Sigma, \Sigma', d, s, \mu)$, where Σ' is an alphabet of *types*, (Σ', d, s) is a DTD($\mathcal{R}$) over Σ', and μ is a mapping from Σ' to Σ.
A tree t then *satisfies* an extended DTD if $t = \mu(t')$ for some $t' \in L(d)$. Here we abuse notation and let μ also denote its extension to define a homomorphism on trees. Again, we denote by $L(D)$ the set of trees satisfying D. For ease of exposition, we always take $\Sigma' = \{\mathsf{a}^i \mid 1 \leqslant i \leqslant k_a,\ a \in \Sigma,\ i \in \mathbb{N}\}$ for some natural numbers k_a, and we set $\mu(\mathsf{a}^i) = a$.
(3) A *single-type EDTD* (EDTD$^{\mathrm{st}}$($\mathcal{R}$)) over Σ is an EDTD($\mathcal{R}$) $D = (\Sigma, \Sigma', d, s, \mu)$ with the property that for every $a \in \Sigma'$, in the regular expression $d(a)$ no two types b^i and b^j with $i \neq j$ occur.

We denote by EDTD, and EDTD$^{\mathrm{st}}$ the classes EDTD(RE), and EDTD$^{\mathrm{st}}$(RE), respectively. As explained in [6,5], EDTDs and single-type EDTDs correspond to Relax NG and XML Schema, respectively. Furthermore, EDTDs correspond to the unranked regular languages [1], while single-type EDTDs form a strict subset thereof [6].

A regular tree language $\mathcal{T}$ is closed under *label-guarded subtree exchange* if it has the following property: if two trees t_1 and t_2 are in $\mathcal{T}$, and there are two nodes v_1 in t_1 and v_2 in t_2 with the same label, then $t_1[v_1 \leftarrow \text{subtree}^{t_2}(v_2)]$ is also in $\mathcal{T}$. This notion is graphically illustrated in Fig. 1.

Lemma 4. *(See [4].) A regular tree language is definable by a DTD iff it is closed under label-guarded subtree exchange.*

An EDTD $D = (\Sigma, \Sigma', d, s_d, \mu)$ is *trimmed* if for every $\mathsf{a}^i \in \Sigma'$, there exists a tree $t \in L(d)$ and a node $u \in \mathrm{Dom}(t)$ such that $\mathrm{lab}^t(u) = \mathsf{a}^i$.

Lemma 5. *(See [6].)*

(1) *For every EDTD D, a trimmed EDTD D', with $L(D) = L(D')$, can be constructed in time polynomial in the size of D.*
(2) *Let D be a trimmed EDTD. For any type $\mathsf{a}^i \in \Sigma'$ and any string $w \in L(d(\mathsf{a}^i))$ there exists a tree $t \in L(d)$ which contains a node v with $\mathrm{lab}^t(v) = \mathsf{a}^i$ and $\text{ch-str}^t(v) = w$.*

We give another schema formalism equivalent to single-type EDTDs. An *automaton-based schema* D over vocabulary Σ is a tuple (A, λ), where $A = (Q, q_0, \delta, F)$ is a DFA and λ is a function mapping states of A to regular expressions. A tree t is accepted by D if for every node v of t, where $q \in Q$ is the state such that $q_0 \Rightarrow_{A, \text{anc-str}(v)} q$, $\text{ch-str}(v) \in L(\lambda(q))$. Because the set of final states F of A is not used, we often omit F and represent A as a triple (Q, q_0, δ).

Remark 6. Because DTDs and EDTDs only define tree languages in which every tree has the same root element, we implicitly assume that this is also the case for automaton-based schemas and the pattern-based schemas defined next. Whenever we

translate among pattern-based schemas, we drop this assumption. Obviously, this does not influence any of the results of this paper.

Lemma 7. *Any automaton-based schema D can be translated into an equivalent single-type EDTD D' in time at most quadratic in the size of D, and vice versa.*

Proof. Let $D = (A, \lambda)$, with $A = (Q, q_0, \delta)$, be an automaton-based schema. We start by making A complete. That is, we add a sink state $q\_$ to Q and for every pair $q \in Q$, $a \in \Sigma$, for which there is no transition $(q, a, q') \in \delta$, we add $(q, a, q\_)$ to δ. Further, $\lambda(q\_) = \emptyset$. Construct $D' = (\Sigma, \Sigma', d, s^i, \mu)$ as follows. Let s^i be such that s is the root symbol of any tree defined by D and $(q_0, s, q_i) \in \delta$. Let $Q \cup \{q\_\} = \{q_0, \ldots, q_n\}$ for some $n \in \mathbb{N}$, then $\Sigma' = \{\mathsf{a}^i \mid a \in \Sigma \wedge q_i \in Q\}$ and $\mu(\mathsf{a}^i) = a$. Finally, $d(\mathsf{a}^i) = \lambda(q_i)$, where any symbol $a \in \Sigma$ is replaced by a^j when $(q_i, a, q_j) \in \delta$. Since A is complete, a^j is guaranteed to exist and since A is a DFA a^j is uniquely defined. For the time complexity of the algorithm, we see that the number of types in D' can never be exceeded by the number of transitions in A. Then, to every type one regular expression from D' is assigned which yields a quadratic algorithm.

Conversely, let $D = (\Sigma, \Sigma', d, s, \mu)$ be a single-type EDTD. The equivalent automaton-based schema $D = (A, \lambda)$ with $A = (Q, q_0, \delta)$ is constructed as follows. Let $Q = \Sigma'$, $q_0 = s$, and for $\mathsf{a}^i, \mathsf{b}^j \in \Sigma'$, $(\mathsf{a}^i, b, \mathsf{b}^j) \in \delta$ if $\mu(\mathsf{b}^j) = b$ and b^j occurs in $d(\mathsf{a}^i)$. Note that since D is a single-type EDTD, A is guaranteed to be deterministic. Finally, for any type $\mathsf{a}^i \in \Sigma'$, $\lambda(\mathsf{a}^i) = \mu(d(\mathsf{a}^i))$. □

2.3. Pattern-based XML schemas

We recycle the following definitions from [7].

Definition 8. A *pattern-based schema* P is a set $\{(r_1, s_1), \ldots, (r_m, s_m)\}$ where all r_i, s_i are regular expressions.

Each pair (r_i, s_i) of a pattern-based schema represents a schema rule. We also refer to the r_i and s_i as the vertical and horizontal regular expressions, respectively. There are two semantics for pattern-based schemas.

Definition 9. A tree t is *existentially valid* with respect to a pattern-based schema P if, for every node v of t, there is a rule $(r, s) \in P$ such that anc-str$(v) \in L(r)$ and ch-str$(v) \in L(s)$. In this case, we write $P \models_\exists t$.

Definition 10. A tree t is *universally valid* with respect to a pattern-based schema P if, for every node v of t, and each rule $(r, s) \in P$ it holds that anc-str$(v) \in L(r)$ implies ch-str$(v) \in L(s)$. In this case, we write $P \models_\forall t$.

Denote by $P_\exists(t) = \{v \in \mathrm{Dom}(t) \mid \exists (r, s) \in P,\ \text{anc-str}(v) \in L(r) \wedge \text{ch-str}(v) \in L(s)\}$ the set of nodes in t that are existentially valid. Denote by $P_\forall(t) = \{v \in \mathrm{Dom}(t) \mid \forall (r, s) \in P,\ \text{anc-str}(v) \in L(r) \Rightarrow \text{ch-str}(v) \in L(s)\}$ the set of nodes in t that are universally valid.

We denote the set of Σ-trees which are existentially and universally valid with respect to P by $\mathcal{T}_\exists^\Sigma(P)$ and $\mathcal{T}_\forall^\Sigma(P)$, respectively. We often Σ if it is clear from the context what the alphabet is.

When for every string $w \in \Sigma^*$ there is a rule $(r, s) \in P$ such that $w \in L(r)$, then we say that P is *complete*. Further, when for every pair $(r, s), (r', s') \in P$ of different rules, $L(r) \cap L(r') = \emptyset$, then we say that P is *disjoint*.

In some proofs, we make use of unary trees, which can be represented as strings. In this context, we abuse notation and write for instance $w \in \mathcal{T}_\exists(P)$ meaning that the unary tree which w represents is existentially valid with respect to P. Similarly, we refer to the last position of w as the leaf of w.

Lemma 11. *For a pattern-based schema P, a tree t and a string w*

(1) *$t \in \mathcal{T}_\forall(P)$ iff for every node v of t, $v \in P_\forall(t)$.*
(2) *If $w \in \mathcal{T}_\forall(P)$ then for every prefix w' of w and every non-leaf node v of w', $v \in P_\forall(w')$.*
(3) *$t \in \mathcal{T}_\exists(P)$ iff for every node v of t, $v \in P_\exists(t)$.*
(4) *If $w \in \mathcal{T}_\exists(P)$ then for every prefix w' of w and every non-leaf node v of w', $v \in P_\exists(w')$.*

Proof. (1), (3) These are in fact just a restatement of the definition of universal and existential satisfaction and are therefore trivially true.

(2) Consider any non-leaf node v' of w'. Since w' is a prefix of w, there must be a node v of w such that anc-str$^w(v) =$ anc-str$^{w'}(v')$ and ch-str$^w(v) =$ ch-str$^{w'}(v')$. By Lemma 11(1), $v \in P_\forall(w)$ and thus $v' \in P_\forall(w)$.

(4) The proof of (2) carries over literally for the existential semantics. □

Lemma 12. *For any complete and disjoint pattern-based schema P, $\mathcal{T}_\exists(P) = \mathcal{T}_\forall(P)$.*

Proof. We show that if P is complete and disjoint, then for any node v of any tree t, $v \in P_\exists(t)$ iff $v \in P_\forall(t)$. The lemma then follows from Lemma 11(1) and (3). First, suppose $v \in P_\exists(t)$. Then, there is a rule $(r, s) \in P$ such that anc-str$(v) \in L(r)$ and ch-str$(v) \in L(s)$, and by the disjointness of P, anc-str$(v) \notin L(r')$ for any other vertical expression r' in P. It thus follows that $v \in P_\forall(t)$. Conversely, suppose $v \in P_\forall(t)$. By the completeness of P there is at least one rule (r, s) such that anc-str$(v) \in L(r)$ and thus ch-str$(v) \in L(s)$. It follows that $v \in P_\exists(t)$. □

2.4. Problems

We give an overview of the problems studied by Schwentick and Kasneci [7] and the ones studied in this paper. We define all problems for the existential semantics, and leave the identical definitions for the universal semantics implicit.

Definition 13. Given pattern-based schemas P, P'

- SATISFIABILITY for P: Is there a non-empty tree t such that $t \in \mathcal{T}_\exists(P)$?
- INCLUSION for P, P': Is $\mathcal{T}_\exists(P) \subseteq \mathcal{T}_\exists(P')$?
- SIMPLIFICATION for P: Does there exist a DTD D with $\mathcal{T}_\exists(P) = L(D)$?

2.5. Succinctness

We introduce some additional notation to characterize the complexity of translating pattern-based schemas into DTDs and (single-type) EDTDs.

For a class $\mathcal{S}$ and $\mathcal{S}'$ of representations of schema languages, and F a class of functions from $\mathbb{N}$ to $\mathbb{N}$, we write $\mathcal{S} \xrightarrow{F} \mathcal{S}'$ if there is an $f \in F$ such that for every $s \in \mathcal{S}$ there is an $s' \in \mathcal{S}'$ with $L(s) = L(s')$ which can be constructed in time $f(|s|)$. This also implies that $|s'| \leqslant f(|s|)$. By $L(s)$ we mean the set of trees defined by s.

We write $\mathcal{S} \overset{F}{\Rightarrow} \mathcal{S}'$ if $\mathcal{S} \xrightarrow{F} \mathcal{S}'$ and there is an $f \in F$, a monotonically increasing function $g : \mathbb{N} \to \mathbb{N}$ and an infinite family of schemas $s_n \in \mathcal{S}$ with $|s_n| \leqslant g(n)$ such that the smallest $s' \in \mathcal{S}'$ with $L(s) = L(s')$ is at least of size $f(g(n))$. By poly, exp and 2-exp we denote the classes of functions $\bigcup_{k,c} cn^k$, $\bigcup_{k,c} c2^{n^k}$ and $\bigcup_{k,c} c2^{2^{n^k}}$, respectively.

Further, we write $\mathcal{S} \nrightarrow \mathcal{S}'$ if there exists an $s \in \mathcal{S}$ such that for every $s' \in \mathcal{S}'$, $L(s') \neq L(s)$. In this case we also write $\mathcal{S} \overset{F}{\nrightarrow} \mathcal{S}'$ and $\mathcal{S} \overset{F}{\not\Rightarrow} \mathcal{S}'$ whenever $\mathcal{S} \xrightarrow{F} \mathcal{S}'$ and $\mathcal{S} \overset{F}{\Rightarrow} \mathcal{S}'$, respectively, hold for those elements in $\mathcal{S}$ which do have an equivalent element in $\mathcal{S}'$.

3. Regular pattern-based schema's

In this section, we study the full class of pattern-based schemas which we denote by $\mathcal{P}_\exists(\text{Reg})$ and $\mathcal{P}_\forall(\text{Reg})$. The results are shown in Theorem 14. Notice that the translations among schemas with different semantics, and the translation from a pattern-based schema under universal semantics to an EDTD are double exponential, whereas the translation from a schema under existential semantics to an EDTD is "only" exponential. Essentially all these double exponential lower bounds are due to the fact that in these translations one necessarily has to apply operations, such as intersection and complement, on regular expressions, which yields double exponential lower bounds. In the translation from a pattern-based schema under existential semantics to an EDTD such operations are not necessary which allows for an easier translation.

Theorem 14.

(1) $\mathcal{P}_\exists(\text{Reg}) \overset{\text{2-exp}}{\Rightarrow} \mathcal{P}_\forall(\text{Reg})$.
(2) $\mathcal{P}_\exists(\text{Reg}) \overset{\text{exp}}{\Rightarrow} \text{EDTD}$.
(3) $\mathcal{P}_\exists(\text{Reg}) \overset{\text{exp}}{\Rightarrow} \text{EDTD}^{\text{st}}$.
(4) SIMPLIFICATION *for* $\mathcal{P}_\exists(\text{Reg})$ *is* EXPTIME*-complete.*
(5) $\mathcal{P}_\exists(\text{Reg}) \overset{\text{exp}}{\nrightarrow} \text{DTD}$.
(6) $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\Rightarrow} \mathcal{P}_\exists(\text{Reg})$.
(7) $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\Rightarrow} \text{EDTD}$.
(8) $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\Rightarrow} \text{EDTD}^{\text{st}}$.
(9) SIMPLIFICATION *for* $\mathcal{P}_\forall(\text{Reg})$ *is* EXPTIME*-complete.*
(10) $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\not\Rightarrow} \text{DTD}$.

Proof. (1) We first show $\mathcal{P}_\exists(\text{Reg}) \overset{\text{2-exp}}{\to} \mathcal{P}_\forall(\text{Reg})$. Let $P = \{(r_1, s_1), \ldots, (r_n, s_n)\}$. We show that we can construct a complete and disjoint pattern-based schema P' such that $\mathcal{T}_\exists(P) = \mathcal{T}_\exists(P')$ in time double exponential in the size of P. By Lemma 12, $\mathcal{T}_\exists(P') = \mathcal{T}_\forall(P')$ and thus $\mathcal{T}_\exists(P) = \mathcal{T}_\forall(P')$.

For any non-empty set $C \subseteq \{1, \ldots, n\}$, denote by r_C the regular expression which defines the language $\bigcap_{i \in C} L(r_i) \setminus \bigcup_{1 \leqslant i \leqslant n,\, i \notin C} L(r_i)$ and by $r_\emptyset$ the expression defining $\Sigma^* \setminus \bigcup_{1 \leqslant i \leqslant n} L(r_i)$. That is, r_C defines any word w which is defined by all vertical expressions contained in C but is not defined by any vertical expression not contained in C. Denote by s_C the expression defining the language $\bigcup_{i \in C} L(s_i)$. Then, $P' = \{(r_\emptyset, \emptyset)\} \cup \{(r_C, s_C) \mid C \subseteq \{1, \ldots, n\} \wedge C \neq \emptyset\}$. Here, P' is disjoint and complete. We show that $\mathcal{T}_\exists(P) = \mathcal{T}_\exists(P')$. By Lemma 11(3), it suffices to prove that for any node v of any tree t, $v \in P_\exists(t)$ iff $v \in P'_\exists(t)$:

- $v \in P_\exists(t) \Rightarrow v \in P'_\exists(t)$: Let $C = \{i \mid \text{anc-str}(v) \in L(r_i)\}$. Since $v \in P_\exists(t)$, $C \neq \emptyset$ and there is an $i \in C$ with $\text{ch-str}(v) \in L(s_i)$. But then, by definition of r_C and s_C, $\text{anc-str}(v) \in L(r_C)$ and $\text{ch-str}(v) \in L(s_C)$, and thus $v \in P'_\exists(t)$.
- $v \in P'_\exists(t) \Rightarrow v \in P_\exists(t)$: Let $C \subseteq \{1, \ldots, n\}$ be the unique set for which $\text{anc-str}(v) \in L(r_C)$ and $\text{ch-str}(v) \in L(s_C)$, and choose some $i \in C$ for which $\text{ch-str}(v) \in L(s_i)$. By definition of s_C, such an i must exist. Then, $\text{anc-str}(v) \in L(r_i)$ and $\text{ch-str}(v) \in L(s_i)$, from which it follows that $v \in P_\exists(t)$.

We conclude by showing that P' can be constructed from P in time double exponential in the size of P. By Lemma 2(1), the expressions r_C can be constructed in time double exponential in the size of the r_i and s_i. The expressions s_C can easily be constructed in linear time by taking the disjunction of the right expressions. So, any rule (r_C, s_C) requires at most double exponential time to construct, and we must construct an exponential number of these rules, which yields and algorithm of double exponential time complexity.

To show that $\mathcal{P}_\exists(\text{Reg}) \overset{\text{2-exp}}{\Rightarrow} \mathcal{P}_\forall(\text{Reg})$, we slightly extend Theorem 2(4).

Lemma 15. *For every $n \in \mathbb{N}$, there is a regular expressions r_n of size linear in n such that any regular expression r defining $\Sigma^* \setminus L(r_n)$ is of size at least double exponential in r. Further, r_n has the property that for any string $w \notin L(r_n)$, there exists a string u such that $wu \in L(r_n)$.*

Proof. Let $n \in \mathbb{N}$. By Theorem 2(4), there exists a regular expression s_n of size linear in n over an alphabet Σ such that any regular expression defining $\Sigma^* \setminus L(s_n)$ must be of size at least double exponential in n. Let $\Sigma_a = \Sigma \uplus \{a\}$, for $a \notin \Sigma$. Define $r_n = s_n + \Sigma_a^* a$ as all strings which are defined by s_n or have a as last symbol. First, note that r_n satisfies the extra condition: for every $w \notin L(r_n)$, $wa \in L(r_n)$. We show that any expression r defining the complement of r_n must be of size at least double exponential in n. This complement consists of all strings which don't have a as last symbol and are not defined by s_n. But then, the expression s which defines $L(r) \cap \Sigma^*$ defines exactly $L(s_n) \setminus \Sigma^*$, the complement of $L(s_n)$. Furthermore, by Theorem 2(4), s must be of size at least double exponential in n and by Theorem 2(5), s can be computed from r in time linear in the size of r. It follows that r must also be of size at least double exponential in n. □

Now, let $n \in \mathbb{N}$ and let r_n be a regular expression over Σ satisfying the conditions of Lemma 15. Then, define $P_n = \{(r_n, \varepsilon), (\Sigma^*, \Sigma)\}$. Here, $\mathcal{T}_\exists(P_n)$ defines all unary trees w for which $w \in L(r_n)$.

Let P be a pattern-based schema with $\mathcal{T}_\exists(P_n) = \mathcal{T}_\forall(P)$. Define $U = \{r \mid (r, s) \in P \wedge \varepsilon \notin L(s)\}$ as the set of vertical regular expressions in P whose corresponding horizontal regular expression does not contain the empty string. Finally, let r be the disjunction of all expressions in U. We now show that $L(r) = \Sigma^* \setminus L(r_n)$, thereby proving that the size of P must be at least double exponential in n.

First, let $w \notin L(r_n)$ and towards a contradiction suppose $w \notin L(r)$. Then, $w \notin \mathcal{T}_\exists(P_n) = \mathcal{T}_\forall(P)$. By Lemma 15, there exists a string u such that $wu \in L(r_n)$, and thus $wu \in \mathcal{T}_\exists(P_n)$ by definition of P_n and so $wu \in \mathcal{T}_\forall(P)$. By Lemma 11(2), for every non-leaf node v of w, $v \in P_\forall(w)$. As w is not defined by any expression in U, for any rule $(r', s') \in P$ with $w \in L(r')$ it holds that $\varepsilon \in L(s')$, and thus for the leaf node v of w, $v \in P_\forall(w)$. So, by Lemma 11(1), $w \in \mathcal{T}_\forall(P)$ which leads to the desired contradiction.

Conversely, suppose $w \in L(r')$, for some $r' \in U$, and again towards a contradiction suppose $w \in L(r_n)$. Then, $w \in \mathcal{T}_\exists(P) = \mathcal{T}_\forall(P)$. But, since $w \in L(r')$, and by definition of U for the rule (r', s') in P it holds that $\varepsilon \notin L(s')$. It follows that the leaf node v of w is not in $P_\forall(w)$. Therefore, $w \notin \mathcal{T}_\forall(P)$ by Lemma 11(1), which again gives us the desired contradiction. This concludes the proof of Theorem 14(6).

(2)–(3) We first show $\mathcal{P}_\exists(\text{Reg}) \overset{\text{exp}}{\to} \text{EDTD}^{\text{st}}$, which implies $\mathcal{P}_\exists(\text{Reg}) \overset{\text{exp}}{\to} \text{EDTD}$.

Thereto, let $P = \{(r_1, s_1), \ldots, (r_n, s_n)\}$. We construct an automaton-based schema $D = (A, \lambda)$ such that $L(D) = \mathcal{T}_\exists(P)$. By Lemma 7, D can then be translated into an equivalent single-type EDTD in polynomial time and the theorem follows. First, construct for every r_i a DFA $A_i = (Q_i, q_i, \delta_i, F_i)$, such that $L(r_i) = L(A_i)$. Then, $A = (Q_1 \times \cdots \times Q_n, (q_1, \ldots, q_n), \delta)$ is the product automaton for $A_1, \ldots, A_n$. Finally, $\lambda((q_1, \ldots, q_n)) = \bigcup_{i \leqslant n, q_i \in F_i} L(s_i)$, and $\lambda((q_1, \ldots, q_n)) = \emptyset$ if none of the q_i are accepting states for their automaton. Here, if m is the size of the largest vertical expression in P, then A is of size $O(2^{m \cdot n})$. Furthermore, an expression for $\bigcup_{i \leqslant n, q_i \in F_i} L(s_i)$ is simply the disjunction of these s_i and can be constructed in linear time. Therefore, the total construction can be carried out in exponential time.

Further, $\mathcal{P}_\exists(\text{Reg}) \overset{\text{exp}}{\Rightarrow} \text{EDTD}$ already holds for a restricted version of pattern-based schemas, which is shown in Theorem 16(2). The latter implies $\mathcal{P}_\exists(\text{Reg}) \overset{\text{exp}}{\Rightarrow} \text{EDTD}^{\text{st}}$.

(4) For the upperbound, we combine a number of results of Kasneci and Schwentick [7] and Martens et al. [6]. In the following, an NTA(NFA) is a non-deterministic tree automaton where the transition relation is represented by an NFA. A DTD(NFA) is a DTD where content models are defined by NFAs.

Given a pattern-based schema P, we first construct an NTA(NFA) A_P with $L(A_P) = \mathcal{T}_\exists(P)$, which can be done in exponential time (Proposition 3.3 in [7]). Then, Martens et al. [6] have shown that given any NTA(NFA) A_P it is possible to construct, in time polynomial in the size of A_P, a DTD(NFA) D_P such that $L(A_P) \subseteq L(D_P)$ is always true and $L(A_P) = L(D_P)$ holds iff $L(A_P)$ is definable by a DTD. Summarizing, D_P is of size exponential in P, $\mathcal{T}_\exists(P) \subseteq L(D_P)$ and $\mathcal{T}_\exists(P)$ is definable by a DTD iff $\mathcal{T}_\exists(P) = L(D_P)$.

Now, construct another NTA(NFA) $A_{\neg P}$ which defines the complement of $\mathcal{T}_\exists(P)$. This can again be done in exponential time (Proposition 3.3 in [7]). Since $\mathcal{T}_\exists(P) \subseteq L(D_P)$, $\mathcal{T}_\exists(P) = L(D_P)$ iff $L(D_P) \cap L(A_{\neg P}) \neq \emptyset$. Here, D_P and $A_{\neg P}$ are of size at most exponential in the size of P, and testing the non-emptiness of their intersection can be done in time polynomial in the size of D_P and $A_{\neg P}$. This gives us an EXPTIME algorithm overall.

For the lower bound, we reduce from SATISFIABILITY of pattern-based schemas, which is EXPTIME-complete [7]. Let P be a pattern-based schema over the alphabet Σ, define $\Sigma^P = \{a, b, c, e\} \uplus \Sigma$, and define the pattern-based schema $P' = \{(a, b + c), (ab, e), (ac, e), (abe, \varepsilon), (ace, \varepsilon)\} \cup \{(acer, s) \mid (r, s) \in P\}$. We show that $\mathcal{T}_\exists(P')$ is definable by a DTD iff P is not existentially satisfiable. Since EXPTIME is closed under complement, the theorem follows.

If $\mathcal{T}_\exists(P) = \emptyset$, then the following DTD d defines $\mathcal{T}_\exists(P')$: $d(a) = b + c$, $d(b) = e$, $d(c) = e$, $d(e) = \varepsilon$.

Conversely, if there exists some tree $t \in \mathcal{T}_\exists(P)$, suppose towards a contradiction that there exists a DTD D such that $L(D) = \mathcal{T}_\exists(P')$. Then, $a(b(e)) \in L(D)$, and $a(c(e(t))) \in L(D)$. Since every DTD is closed under label-guarded subtree exchange (Lemma 4), $a(b(e(t))) \in L(D)$ also holds, but $a(b(e(t))) \notin \mathcal{T}_\exists(P')$ which yields the desired contradiction.

(5) First, $\mathcal{P}_\exists(\text{Reg}) \nrightarrow \text{DTD}$ already holds for a restricted version of pattern-based schemas (Theorem 20(6)). We show $\mathcal{P}_\exists(\text{Reg}) \overset{\text{exp}}{\nrightarrow} \text{DTD}$.

Simply translating the DTD(NFA), obtained in the previous proof, into a normal DTD by means of state elimination would give us a double exponential algorithm. Therefore, we use the following similar approach which does not need to translate regular expressions into NFAs and back. First, construct a single-type EDTD D_1 such that $L(D_1) = \mathcal{T}_\exists(P)$. This can be done in exponential time according to Theorem 14(3). Then, use the polynomial time algorithm of Martens et al. [6], to construct an equivalent DTD D. In this algorithm, all expressions of D define unions of the language defined by the expressions in D_1. This can, of course, be done by taking the disjunction of expressions in D_1. In total, D is constructed in exponential time.

(6) We first show $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\to} \mathcal{P}_\exists(\text{Reg})$. We take the same approach as in the proof of Theorem 14(1), but have to make some small changes. Let $P = \{(r_1, s_1), \ldots, (r_n, s_n)\}$, and for any non-empty set $C \subseteq \{1, \ldots, n\}$ let r_C be the regular expression defining $\bigcap_{i \in C} L(r_i) \setminus \bigcup_{1 \leqslant i \leqslant n, i \notin C} L(r_i)$. Let $r_\emptyset$ define $\Sigma^* \setminus \bigcap_{i \leqslant n} L(r_i)$ and let s_C be the expression defining the language $\bigcap_{i \in C} L(s_i)$. Define $P' = \{(r_\emptyset, \Sigma^*)\} \cup \{(r_C, s_C) \mid C \subseteq \{1, \ldots, n\} \wedge C \neq \emptyset\}$. Here, P' is disjoint and complete and, by the same argumentation as in the proof of Theorem 14(1), can be constructed in time double exponential in the size of P'. So, by Lemma 12, $\mathcal{T}_\exists(P') = \mathcal{T}_\forall(P')$. We show that $\mathcal{T}_\forall(P) = \mathcal{T}_\forall(P')$ from which $\mathcal{T}_\forall(P) = \mathcal{T}_\exists(P')$ then follows. By Lemma 11(1), it suffices to prove that for any node v of any tree t, $v \in P_\forall(t)$ iff $v \in P'_\forall(t)$:

- $v \in P_\forall(t) \Rightarrow v \in P'_\forall(t)$: Let $C = \{i \mid \text{anc-str}(v) \in L(r_i)\}$. If $C = \emptyset$, then $\text{anc-str}(v) \in r_\emptyset$ and the horizontal regular expression Σ^* allows every child-string. Because of the disjointness of P' no other vertical regular expression in P' can define $\text{anc-str}(v)$ and thus $v \in P'_\forall(t)$. If $C \neq \emptyset$, since $v \in P_\forall(t)$, for all $i \in C$, $\text{ch-str}(v) \in L(s_i)$. But then, by definition of r_C and s_C, $\text{anc-str}(v) \in L(r_C)$ and $\text{ch-str}(v) \in L(s_C)$, combined with the disjointness of P' gives $v \in P'_\forall(t)$.
- $v \in P'_\forall(t) \Rightarrow v \in P_\forall(t)$: Let $C \subseteq \{1, \ldots, n\}$ be the unique set for which $(r_C, s_C) \in P'$, $\text{anc-str}(v) \in L(r_C)$ and $\text{ch-str}(v) \in L(s_C)$. Since $v \in P'_\forall(t)$ and by the disjointness and completeness of P' there indeed exists exactly one such set. If $C = \emptyset$, then $\text{anc-str}(v)$ is not defined by any vertical expression in P and thus $v \in P_\forall(t)$. If $C \neq \emptyset$, then for all $i \in C$, $\text{anc-str}(v) \in L(r_i)$ and $\text{ch-str}(v) \in L(s_i)$, and for all $i \notin C$, $\text{anc-str}(v) \notin L(r_i)$. It follows that $v \in P_\forall(t)$.

We now show that $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\Rightarrow} \mathcal{P}_\exists(\text{Reg})$. Let $n \in \mathbb{N}$. According to Theorem 2(2), there exist a linear number of regular expressions $r_1, \ldots, r_m$ of size linear in n such that any regular expression defining $\bigcap_{i \leqslant m} L(r_i)$ must be of size at least double exponential in n. For brevity, define $K = \bigcap_{i \leqslant m} L(r_i)$.

Define P_n over the alphabet $\Sigma_a = \Sigma \uplus \{a\}$, for $a \notin \Sigma$, as $P_n = \{(a, r_i) \mid i \leqslant m\} \cup \{(ab, \varepsilon) \mid b \in \Sigma\} \cup \{(b, \emptyset) \mid b \in \Sigma\}$. That is, $\mathcal{T}_\forall(P_n)$ contains all trees $a(w)$, where $w \in K$.

Let P be a pattern-based schema with $\mathcal{T}_\forall(P_n) = \mathcal{T}_\exists(P)$. For an expression s, denote by s^- the expression defining all words in $L(s) \cap \Sigma^*$. According to Theorem 2(5), s^- can be constructed from s in linear time. Define $U = \{s^- \mid (r, s) \in P \wedge a \in L(r)\}$ as the set of horizontal regular expressions whose corresponding vertical regular expressions contains the string a. Finally, let r_K be the disjunction of all expressions in U. We now show that $L(r_K) = K$, thereby proving that the size of P must be at least double exponential in n.

First, let $w \in K$. Then, $t = a(w) \in \mathcal{T}_\forall(P_n) = \mathcal{T}_\exists(P)$. Therefore, by Lemma 11(3), the root node v of t is in $P_\exists(t)$. It follows that there must be a rule $(r, s) \in P$, with $a \in L(r)$ and $w \in L(s)$. Now $w \in \Sigma^*$ implies $w \in L(s^-)$, and thus, by definition of U and r_K, $w \in L(r_K)$.

Conversely, suppose $w \in L(s^-)$ for some $s^- \in U$. We show that $t = a(w) \in \mathcal{T}_\exists(P) = \mathcal{T}_\forall(P_n)$, which implies that $w \in K$. By Lemma 11(3), it suffices to show that every node v of t is in $P_\exists(t)$. For the root node v of t, we know that $\text{ch-str}(v) = w \in$

$L(s^-)$, and by definition of U, that anc-str$(v) = a \in L(r)$, where r is the corresponding vertical expression for s. Therefore, $v \in P_\exists(t)$. All other nodes v are leaf nodes with ch-str$(v) = \varepsilon$ and anc-str$(v) = ab$, where $b \in \Sigma$ since $w \in L(s^-)$. To show that any node with these child and ancestor-strings must be in $P_\exists(t)$, note that for every symbol $b \in \Sigma$ there exists a string $w' \in K$ such that w' contains a b. Otherwise b is useless and can be removed from Σ. Then, $t' = a(w') \in \mathcal{T}_\forall(P_n) = \mathcal{T}_\exists(P)$ and thus there is a leaf node v' in t' for which anc-str$(v') = ab$ and ch-str$(v') = \varepsilon$. Since, by Lemma 11(3) $v' \in P_\exists(t')$, also any leaf node v of t with anc-str$(v) = ab$ is in $P_\exists(t)$. It follows that $t \in \mathcal{T}_\exists(P) = \mathcal{T}_\forall(P_n)$.

(7)–(8) We first show $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\to} \text{EDTD}^{\text{st}}$, which implies $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\to} \text{EDTD}$. Thereto, let $P = \{(r_1, s_1), \ldots, (r_n, s_n)\}$. We construct an automaton-based schema $D = (A, \lambda)$ such that $L(D) = \mathcal{T}_\forall(P)$. By Lemma 7, D can then be translated into an equivalent single-type EDTD and the theorem follows. We construct A in exactly the same manner as in the proof of Theorem 14(3). For λ, let $\lambda((q_1, \ldots, q_n)) = \bigcap_{i \leqslant n, q_i \in F_i} L(s_i)$, and $\lambda((q_1, \ldots, q_n)) = \Sigma^*$ if none of the q_i are accepting states for their automaton. We already know that A can be constructed in exponential time, and by Theorem 2(2) a regular expression for $\lambda((q_1, \ldots, q_n)) = \bigcap_{i \leqslant n, q_i \in F_i} L(s_i)$ can be constructed in double exponential time. It follows that the total construction can be done in double exponential time.

Further, $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\Rightarrow} \text{EDTD}$ already holds for a restricted version of pattern-based schemas, which is shown in Theorem 16(7). The latter implies $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\Rightarrow} \text{EDTD}^{\text{st}}$.

(9) The proof is along the same lines as that of Theorem 14(4).

(10) First, $\mathcal{P}_\forall(\text{Reg}) \nrightarrow \text{DTD}$ already holds for a restricted version of pattern-based schemas (Theorem 20(12)).

We first show $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\nrightarrow} \text{DTD}$. Notice that the DTD(NFA) D constructed in the above proof, conform the proof of Theorem 14(4), is constructed in time exponential in the size of P. To obtain an actual DTD, we only have to translate the NFAs in D into regular expressions, which can be done in exponential time by means of state elimination. This yields a total algorithm of double exponential time complexity.

Finally, $\mathcal{P}_\forall(\text{Reg}) \overset{\text{2-exp}}{\not\Rightarrow} \text{DTD}$ already holds for a more restricted version of pattern-based schemas, which is shown in Theorem 16(10). □

4. Linear pattern-based schemas

In this section, following [7], we restrict the vertical expressions to XPath expressions using only descendant and child axes. For instance, an XPath expression $\backslash\backslash a \backslash\backslash b \backslash c$ captures all nodes that are labeled with c, have b as parent and have an a as ancestor. This corresponds to the regular expression $\Sigma^* a \Sigma^* bc$.

Formally, we call an expression *linear* if it is of the form $w_0 \Sigma^* \cdots w_{n-1} \Sigma^* w_n$, with $w_0, w_n \in \Sigma^*$, and $w_i \in \Sigma^+$ for $1 \leqslant i < n$. A pattern-based schema is *linear* if all its vertical expressions are linear. Denote the classes of linear schemas under existential and universal semantics by $\mathcal{P}_\exists(\text{Lin})$ and $\mathcal{P}_\forall(\text{Lin})$, respectively.

Theorem 16 lists the results for linear schemas. The complexity of SIMPLIFICATION improves slightly, PSPACE instead of EXPTIME. Further, we show that the expressive power of linear schemas under existential and universal semantics becomes incomparable, but that the complexity of translating to DTDs and (single-type) EDTDs is in general not better than for regular pattern-based schemas.

Theorem 16.

(1) $\mathcal{P}_\exists(\text{Lin}) \nrightarrow \mathcal{P}_\forall(\text{Lin})$.
(2) $\mathcal{P}_\exists(\text{Lin}) \overset{\text{exp}}{\Rightarrow} \text{EDTD}$.
(3) $\mathcal{P}_\exists(\text{Lin}) \overset{\text{exp}}{\Rightarrow} \text{EDTD}^{\text{st}}$.
(4) SIMPLIFICATION *for* $\mathcal{P}_\exists(\text{Lin})$ *is* PSPACE*-complete.*
(5) $\mathcal{P}_\exists(\text{Lin}) \overset{\text{exp}}{\nrightarrow} \text{DTD}$.
(6) $\mathcal{P}_\forall(\text{Lin}) \nrightarrow \mathcal{P}_\exists(\text{Lin})$.
(7) $\mathcal{P}_\forall(\text{Lin}) \overset{\text{2-exp}}{\Rightarrow} \text{EDTD}$.
(8) $\mathcal{P}_\forall(\text{Lin}) \overset{\text{2-exp}}{\Rightarrow} \text{EDTD}^{\text{st}}$.
(9) SIMPLIFICATION *for* $\mathcal{P}_\forall(\text{Lin})$ *is* PSPACE*-complete.*
(10) $\mathcal{P}_\forall(\text{Lin}) \overset{\text{2-exp}}{\not\Rightarrow} \text{DTD}$.

Proof. (1) We first prove the following simple lemma. Given an alphabet Σ, and a symbol $b \in \Sigma$, denote $\Sigma \setminus \{b\}$ by Σ_b.

Lemma 17. *There does not exist a set of linear regular expression $r_1, \ldots, r_n$ such that $\bigcup_{1 \leqslant i \leqslant n} L(r_i)$ is an infinite language and $\bigcup_{1 \leqslant i \leqslant n} L(r_i) \subseteq L(\Sigma_b^*)$.*

Proof. Suppose to the contrary that such a list of linear expressions does exist. Then, one of these expressions must contain Σ^* because otherwise $\bigcup_{1 \leqslant i \leqslant n} L(r_i)$ would be a finite language. However, if an expression contains Σ^*, then it also defines words containing b, which gives us the desired contradiction. □

Now, let $P = \{(\Sigma^* b \Sigma^*, \varepsilon), (\Sigma^*, \Sigma)\}$. Then, $\mathcal{T}_\exists(P)$ defines all unary trees containing at least one b. Suppose that P' is a linear schema such that $\mathcal{T}_\exists(P) = \mathcal{T}_\forall(P')$. Define $U = \{r \mid (r, s) \in P' \text{ and } \varepsilon \notin L(s)\}$ as the set of all vertical regular expressions in P' whose horizontal regular expressions do not contain the empty string. We show that the union of the expressions in U defines an infinite language and is a subset of Σ_b^*, which by Lemma 17 proves that such a schema P' cannot exist.

First, to show that the union of these expressions defines an infinite language, suppose that it does not. Then, every expression $r \in U$ is of the form $r = w$, for some string w. Let k be the length of the longest such string w. Now, $a^{k+1}b \in \mathcal{T}_\exists(P) = \mathcal{T}_\forall(P')$ and thus by Lemma 11(2) every non-leaf node v of a^{k+1} is in $P'_\forall(a^{k+1})$. Further, $a^{k+1} \notin L(r)$ for all vertical expressions in U and thus the leaf node of a^{k+1} is also in $P'_\forall(a^{k+1})$. But then, by Lemma 11(1), $a^{k+1} \in \mathcal{T}_\forall(P')$ which leads to the desired contradiction.

Second, let $w \in L(r)$, for some $r \in U$, we show $w \in \Sigma_b^*$. Towards a contradiction, suppose $w \notin \Sigma_b^*$, which means that w contains at least one b and thus $w \in \mathcal{T}_\exists(P) = \mathcal{T}_\forall(P')$. But then, for the leaf node v of w, anc-str$(v) = w \in L(r)$, and by definition of U, ch-str$(v) = \varepsilon \notin L(s)$, where s is the corresponding horizontal expression for r. Then, $v \notin P'_\forall(w)$ and thus by Lemma 11(1), $w \notin \mathcal{T}_\forall(P')$, which again gives the desired contradiction.

(2)–(3) First, $\mathcal{P}_\exists(\text{Lin}) \overset{\text{exp}}{\to} \text{EDTD}^{\text{st}}$ follows immediately from Theorem 14(3). We show $\mathcal{P}_\exists(\text{Lin}) \overset{\text{exp}}{\Rightarrow} \text{EDTD}$, which then implies both statements. Thereto, we first characterize the expressive power of EDTDs over unary tree languages.

Lemma 18. *For any EDTD D for which $L(D)$ is a unary tree language, there exists an NFA A such that $L(D) = L(A)$. Moreover, A can be computed from D in time linear in the size of D.*

Proof. Let $D = (\Sigma, \Sigma', d, s, \mu)$ be an EDTD, such that $L(D)$ is a unary tree language. Then, define $A = (Q, q_0, \delta, F)$ as $Q = \{q_0\} \cup \Sigma'$, $\delta = \{(q_0, s, s)\} \cup \{(a, \mu(b), b) \mid a, b \in \Sigma' \wedge b \in L(d(a))\}$, and $F = \{a \mid a \in \Sigma' \wedge \varepsilon \in d(a)\}$. □

Now, let $n \in \mathbb{N}$. Define $\Sigma_n = \{\$, \#_1, \#_2\} \cup \bigcup_{1 \leqslant i \leqslant n} \{a_i^0, a_i^1, b_i^0, b_i^1\}$ and $K_n = \{\#_1 a_1^{i_1} a_2^{i_2} \cdots a_n^{i_n} \$ b_1^{i_1} b_2^{i_2} \cdots b_n^{i_n} \#_2 \mid i_k \in \{0, 1\},\ 1 \leqslant k \leqslant n\}$. It is not hard to see that any NFA defining K_n must be of size at least exponential in n. Indeed, in Theorem 1, define $M = \{(x, w) \mid xw \in K_n \wedge |x| = n + 1\}$ which is of size exponential in n, and satisfies the conditions of Theorem 1. Then, by Lemma 18, every EDTD defining the unary tree language K_n must also be of size exponential in n. We conclude the proof by giving a pattern-based schema P_n, such that $\mathcal{T}_\exists(P_n) = K_n$, which is of size linear in n. It contains the following rules:

- $\#_1 \to a_1^0 + a_1^1$
- For any $i < n$:
  - $\#_1 \Sigma^* a_i^0 \to a_{i+1}^0 + a_{i+1}^1$
  - $\#_1 \Sigma^* a_i^1 \to a_{i+1}^0 + a_{i+1}^1$
  - $\#_1 \Sigma^* a_i^0 \Sigma^* b_i^0 \to b_{i+1}^0 + b_{i+1}^1$
  - $\#_1 \Sigma^* a_i^1 \Sigma^* b_i^1 \to b_{i+1}^0 + b_{i+1}^1$
- $\#_1 \Sigma^* a_n^0 \to \$$
- $\#_1 \Sigma^* a_n^1 \to \$$
- $\#_1 \Sigma^* \$ \to b_1^0 + b_1^1$
- $\#_1 \Sigma^* a_n^0 \Sigma^* b_n^0 \to \#_2$
- $\#_1 \Sigma^* a_n^1 \Sigma^* b_n^1 \to \#_2$
- $\#_1 \Sigma^* \#_2 \to \varepsilon$

(4) For the lower bound, we reduce from UNIVERSALITY of regular expressions. That is, deciding for a regular expression r whether $L(r) = \Sigma^*$. The latter problem is known to be PSPACE-complete [11]. Given r over alphabet Σ, let $\Sigma^P = \{a, b, c, d\} \uplus \Sigma$, and define the pattern-based schema $P = \{(a, b + c), (ab, e), (ac, e), (abe, \Sigma^*), (ace, r)\} \cup \{(abe\sigma, \varepsilon), (ace\sigma, \varepsilon) \mid \sigma \in \Sigma\}$. We show that there exists a DTD D with $L(D) = \mathcal{T}_\exists(P)$ iff $L(r) = \Sigma^*$.

If $L(r) = \Sigma^*$, then the following DTD d defines $\mathcal{T}_\exists(P)$: $d(a) = b + c$, $d(b) = e$, $d(c) = e$, $d(e) = \Sigma^*$, and $d(\sigma) = \varepsilon$ for every $\sigma \in \Sigma$.

Conversely, if $L(r) \subsetneq \Sigma^*$, we show that $\mathcal{T}_\exists(P)$ is not closed under label-guarded subtree exchange. From Lemma 4, it then follows that $\mathcal{T}_\exists(P)$ is not definable by a DTD. Let w, w' be strings such that $w \notin L(r)$ and $w' \in L(r)$. Then, $a(b(e(w))) \in L(D)$, and $a(c(e(w'))) \in L(D)$ but $a(c(e(w))) \notin \mathcal{T}_\exists(P)$.

For the upper bound, we again make use of the closure under label-guarded subtree exchange property of DTDs. Observe that $\mathcal{T}_\exists(P)$, which is a regular tree language, is not definable by any DTD iff there exist trees $t_1, t_2 \in \mathcal{T}_\exists(P)$ and nodes v_1 and v_2 in t_1 and t_2, respectively, with $\text{lab}^{t_1}(v_1) = \text{lab}^{t_2}(v_2)$, such that the tree $t_3 = t_1[v_1 \leftarrow \text{subtree}^{t_2}(v_2)]$ is not in $\mathcal{T}_\exists(P)$. We refer to such a tuple (t_1, t_2) as a witness to the DTD-undefinability of $\mathcal{T}_\exists(P)$, or simply a *witness tuple*.

Lemma 19. *If there exists a witness tuple (t_1, t_2) for a linear schema P, then there also exists a witness tuple (t'_1, t'_2) for P, where t'_1 and t'_2 are of depth polynomial in the size of P.*

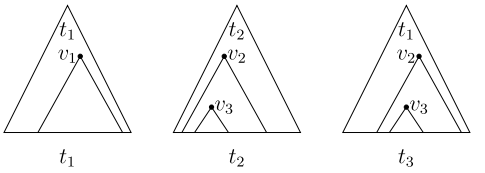

Fig. 2. The five different areas in t_1 and t_2.

Proof. We make use of techniques introduced by Kasneci and Schwentick [7]. When P, P' are two linear schemas, they stated that if there exists a tree t with $t \in \mathcal{T}_\exists(P)$ but $t \notin \mathcal{T}_\exists(P')$, then there exists a tree t' of depth polynomial with the same properties. In particular, they obtained the following property.

Let P be a linear pattern-based schema and t a tree. Then, to every node v of t, a vector $F_P^t(v)$ over $\mathbb{N}$ can be assigned with the following properties:

- along a path in a tree, $F_P^t(v)$ can take at most polynomially many values in the size of P;
- if v' is a child of v, then $F_P^t(v')$ can be computed from $F_P^t(v)$ and the label of v' in t; and
- $v \in P_\exists(t)$ can be decided solely on the value of $F_P^t(v)$ and ch-str(v).

Based on these properties it is easy to see that if there exists a tree t which existentially satisfies P, then there exists a tree t' of polynomial depth which existentially satisfies P. Indeed, t' can be constructed from t by searching for nodes v and v' of t such that v' is a descendant of v, $\text{lab}^t(v) = \text{lab}^t(v')$ and $F_P^t(v) = F_P^t(v')$, and replacing the subtree rooted at v by the one rooted at v'. By applying this rule as often as possible, we get a tree which is still existentially valid with respect to P and where no two nodes on a path in the tree have the same vector and label and which thus is of polynomial depth.

We will also use this technique, but have to be a bit more careful in the replacements we carry out. Thereto, let (t_1, t_2) be a witness tuple for P and fix nodes v_1 and v_2 of t_1 and t_2, respectively, such that t_3, defined as $t_1[v_1 \leftarrow \text{subtree}^{t_2}(v_2)]$, is not in $\mathcal{T}_\exists(P)$. Since $t_3 \notin \mathcal{T}_\exists(P)$, by Lemma 11(3), there must be some node v_3 of t_3 with $v_3 \notin P_\exists(t_3)$. Furthermore, v_3 must occur in the subtree under v_2 inherited from t_2. Indeed, every node v not in that subtree, has the same vector and child-string as its corresponding node in t_1, and since $t_1 \in \mathcal{T}_\exists(P)$ also $v \in P_\exists(t_1)$ and thus $v \in P_\exists(t_3)$. So, fix some node v_3, with $v_3 \notin P_\exists(t_3)$, occurring in t_2. Then, we can partition the trees t_1 and t_2, and thereby also t_3, in five different parts as follows:

(1) $t_1[v_1 \leftarrow ()]$: the tree t_1 without the subtree under v_1;
(2) $\text{subtree}^{t_1}(v_1)$: the subtree under v_1 in t_1;
(3) $t_2[v_2 \leftarrow ()]$: the tree t_2 without the subtree under v_2;
(4) $\text{subtree}^{t_2}(v_2)[v_3 \leftarrow ()]$: the subtree under v_2 in t_2, without the subtree under v_3;
(5) $\text{subtree}^{t_2}(v_3)$: the subtree under v_3 in t_2.

This situation is graphically illustrated in Fig. 2.

Now, let t_1' and t_2' be the trees obtained from t_1 and t_2 by repeating the following as often as possible: Search for two nodes v, v' such that v is an ancestor of v', v and v' are not equal to v_1, v_2 or v_3, v and v' occur in the same part of t_1 or t_2, $\text{lab}(v) = \text{lab}(v')$ and $F_P^{t_1}(v) = F_P^{t_1}(v')$ (or $F_P^{t_2}(v) = F_P^{t_2}(v')$ if v and v' both occur in t_2). Then, replace v by the subtree under v'.

Observe that, by the properties of F, any path in one of the five parts of t_1' and t_2' can have at most a polynomial depth, and thus t_1' and t_2' are of at most a polynomial depth. Furthermore, $t_1', t_2' \in \mathcal{T}_\exists(P)$ still holds and the original nodes v_1, v_2 and v_3 still occur in t_1' and t_2'. Therefore, for $t_3' = t_1'[v_1 \leftarrow \text{subtree}^{t_2'}(v_2)]$, $F_P^{t_3'}(v_3) = F_P^{t_3}(v_3)$ and $\text{ch-str}^{t_3'}(v_3) = \text{ch-str}^{t_3}(v_3)$. But then, $v_3 \notin P_\exists(t_3')$, which by Lemma 11(3) gives us $t_3' \notin \mathcal{T}_\exists(P)$. So, (t_1', t_2') is a witness tuple in which t_1' and t_2' are of at most polynomial depth. □

Now, using Lemma 19, we show that the problem is in PSPACE. We simply guess a witness tuple (t_1, t_2) and check in PSPACE, whether it is a valid witness tuple. If it is, $\mathcal{T}_\exists(P)$ is not definable by a DTD. If $\mathcal{T}_\exists(P)$ is definable by a DTD, there does not exist a witness tuple for P. Since, PSPACE is closed under complement, the theorem follows.

By Lemma 19, it suffices to guess trees of at most polynomial depth. Therefore, we guess t_1 and t_2 in depth-first and left-to-right fashion, maintaining for each tree and each level of the trees, the sets of states the appropriate automata can be in. Here, t_1 and t_2 are guessed simultaneously and independently. That is, for each guessed symbol, we also guess whether it belongs to t_1 or t_2. At some point in this procedure, we guess that we are now at the nodes v_1 and v_2 of t_1 and t_2. From that point we maintain a third list of states of automata, which are initiated by the values of these of t_1, but the subsequent subtree take the values of t_2. If in the end, t_1 and t_2 are accepted, but the third tree is not, then (t_1, t_2) is a valid witness for P.

(5) First, $\mathcal{P}_\exists(\text{Lin}) \nrightarrow \text{DTD}$ already holds for a restricted version of pattern-based schemas (Theorem 20(6)). Then, $\mathcal{P}_\exists(\text{Lin}) \overset{\text{exp}}{\nrightarrow} \text{DTD}$ follows immediately from Theorem 14(5).

(6) Let $\Sigma = \{a, b, c\}$ and define $P = \{(\Sigma^* b \Sigma^* c, b)\}$. Then, $\mathcal{T}_\forall(P)$ contains all trees in which whenever a c labeled node v has a b labeled node as ancestor, ch-str(v) must be b. We show that any linear schema P' defining all trees in $\mathcal{T}_\forall(P)$ under existential semantics, must also define trees not in $\mathcal{T}_\forall(P)$.

Suppose there does exist a linear schema P' such that $\mathcal{T}_\forall(P) = \mathcal{T}_\exists(P')$. Define $w_\ell = a^\ell c$ for $\ell \geqslant 1$ and note that $w_\ell \in \mathcal{T}_\forall(P) = \mathcal{T}_\exists(P')$. Let $(r, s) \in P'$ be a rule matching infinitely many leaf nodes of the strings w_ℓ. There must be at least one as P' contains a finite number of rules. Then, $\varepsilon \in L(s)$ must hold and r is of one of the following forms:

(1) $a^{n_1} \Sigma^* a^{n_2} \Sigma^* \cdots \Sigma^* a^{n_k} c$,
(2) $a^{n_1} \Sigma^* a^{n_2} \Sigma^* \cdots \Sigma^* a^{n_k} c \Sigma^*$,
(3) $a^{n_1} \Sigma^* a^{n_2} \Sigma^* \cdots \Sigma^* a^{n_k} \Sigma^*$,

where $k \geqslant 2$ and $n_k \geqslant 0$.

Choose some $N \in \mathbb{N}$ with $N \geqslant |P'|$ and define the unary trees $t_1 = a^N b a^N c b$ and $t_2 = a^N b a^N c$. Obviously, $t_1 \in \mathcal{T}_\forall(P)$, and $t_2 \notin \mathcal{T}_\forall(P)$. Then, $t_1 \in \mathcal{T}_\exists(P')$ and since t_2 is a prefix of t_1, by Lemma 11(4), every non-leaf node v of t_2 is in $P'_\exists(t_2)$. Finally, for the leaf node v of t_2, anc-str(v) $\in L(r)$ for any of the three expressions given above and $\varepsilon \in L(s)$ for its corresponding horizontal expression. Then, $v \in P'_\exists(t_2)$, and thus by Lemma 11(3), $t_2 \in \mathcal{T}_\exists(P')$ which completes the proof.

(7)–(8) First, $\mathcal{P}_\forall(\text{Lin}) \overset{\text{2-exp}}{\to} \text{EDTD}^{\text{st}}$ follows immediately from Theorem 14(3). We show $\mathcal{P}_\forall(\text{Lin}) \overset{\text{2-exp}}{\Rightarrow} \text{EDTD}$, which then implies both statements.

Let $n \in \mathbb{N}$. According to Theorem 2(3), there exist a linear number of regular expressions $r_1, \ldots, r_m$ of size linear in n such that any regular expression defining $\bigcap_{i \leqslant m} L(r_i)$ must be of size at least double exponential in n. Set $K = \bigcap_{i \leqslant m} L(r_i)$.

Next, we define P_n over the alphabet $\Sigma \uplus \{a\}$ as $P_n = \{(a, r_i) \mid i \leqslant m\} \cup \{(ab, \varepsilon) \mid b \in \Sigma\} \cup \{(b, \emptyset) \mid b \in \Sigma\}$. That is, $\mathcal{T}_\forall(P_n)$ defines all trees $a(w)$, for which $w \in K$.

Let $D = (\Sigma, \Sigma', d, a, \mu)$ be any EDTD with $\mathcal{T}_\forall(P) = L(D)$. By Lemma 5(a), we can assume that D is trimmed. Let $a \to r$ be the single rule in D for the root element a. Let r_K be the expressions defining $\mu(L(r))$. Since D is trimmed, it follows from Lemma 5(2) that r_K cannot contain an a. But then, $L(r_K) = K$, which proves that the size of D must be at least double exponential in n.

(9) The proof is along the same lines as that of Theorem 16(4).

(10) First, $\mathcal{P}_\forall(\text{Lin}) \nrightarrow \text{DTD}$ already holds for a restricted version of pattern-based schemas (Theorem 20(12)). Then, $\mathcal{P}_\forall(\text{Lin}) \overset{\text{2-exp}}{\nrightarrow} \text{DTD}$ follows immediately from Theorem 14(10). For $\mathcal{P}_\forall(\text{Lin}) \overset{\text{2-exp}}{\not\Rightarrow} \text{DTD}$, let $n \in \mathbb{N}$. In the proof of Theorem 16(7) we have defined a linear pattern-based schema P_n of size polynomial in n for which any EDTD D' with $\mathcal{T}_\forall(P_n) = L(D')$ must be of size at least double exponential in n. Furthermore, every DTD is an EDTD and the language $\mathcal{T}_\forall(P_n)$ is definable by a DTD. It follows that any DTD D with $\mathcal{T}_\forall(P_n) = L(D)$ must be of size at least double exponential in n. □

5. Strongly linear pattern-based schemas

In [6], it is observed that the type of a node in most real-world XSDs only depends on the labels of its parents and grand parents. To capture this idea, following [7], we say that a regular expression is *strongly linear* if it is of the form w or $\Sigma^* w$, where w is non-empty. A pattern-based schema is *strongly linear* if it is disjoint and all its vertical expressions are strongly linear. Denote the class of all strongly linear pattern-based schemas under existential and universal semantics by $\mathcal{P}_\exists$(S-Lin) and $\mathcal{P}_\forall$(S-Lin), respectively.

In [7], all horizontal expressions in a strongly linear schema are also required to be deterministic or one-unambiguous [12], as is the case for DTDs and XML Schema. The latter requirement is necessary to get PTIME SATISFIABILITY and INCLUSION which would otherwise be PSPACE-complete for arbitrary regular expressions. This is also the case for the SIMPLIFICATION problem studied here, but not for the various translation problems. Therefore, we distinguish between strongly linear schemas, as defined above, and strongly linear schemas where all horizontal expressions must be deterministic, which we call *deterministic strongly linear schemas* and denote by $\mathcal{P}_\exists$(Det-S-Lin) and $\mathcal{P}_\forall$(Det-S-Lin).

Theorem 20 shows the results for (deterministic) strongly linear pattern-based schemas. First, observe that the expressive power of these schemas under existential and universal semantics again coincides. Further, all considered problems become tractable, which makes strongly linear schemas very interesting from a practical point of view.

Theorem 20.

(1) $\mathcal{P}_\exists(\text{S-Lin}) \overset{\text{poly}}{\to} \mathcal{P}_\forall(\text{S-Lin})$ *and* $\mathcal{P}_\exists(\text{Det-S-Lin}) \overset{\text{poly}}{\to} \mathcal{P}_\forall(\text{Det-S-Lin})$.
(2) $\mathcal{P}_\exists(\text{S-Lin}) \overset{\text{poly}}{\to} \text{EDTD}$ *and* $\mathcal{P}_\exists(\text{Det-S-Lin}) \overset{\text{poly}}{\to} \text{EDTD}$.
(3) $\mathcal{P}_\exists(\text{S-Lin}) \overset{\text{poly}}{\to} \text{EDTD}^{\text{st}}$ *and* $\mathcal{P}_\exists(\text{Det-S-Lin}) \overset{\text{poly}}{\to} \text{EDTD}^{\text{st}}$.
(4) SIMPLIFICATION *for* $\mathcal{P}_\exists$(S-Lin) *is* PSPACE*-complete.*
(5) SIMPLIFICATION *for* $\mathcal{P}_\exists$(Det-S-Lin) *is in* PTIME.
(6) $\mathcal{P}_\exists(\text{S-Lin}) \overset{\text{poly}}{\nrightarrow} \text{DTD}$ *and* $\mathcal{P}_\exists(\text{Det-S-Lin}) \overset{\text{poly}}{\nrightarrow} \text{DTD}$.

(7) $\mathcal{P}_\forall(\text{S-Lin}) \overset{\text{poly}}{\to} \mathcal{P}_\exists(\text{S-Lin})$ *and* $\mathcal{P}_\forall(\text{Det-S-Lin}) \overset{\text{poly}}{\to} \mathcal{P}_\exists(\text{Det-S-Lin})$.

(8) $\mathcal{P}_\forall(\text{S-Lin}) \overset{\text{poly}}{\to} \text{EDTD}$ *and* $\mathcal{P}_\forall(\text{Det-S-Lin}) \overset{\text{poly}}{\to} \text{EDTD}$.

(9) $\mathcal{P}_\forall(\text{S-Lin}) \overset{\text{poly}}{\to} \text{EDTD}^{\text{st}}$ *and* $\mathcal{P}_\forall(\text{Det-S-Lin}) \overset{\text{poly}}{\to} \text{EDTD}^{\text{st}}$.

(10) SIMPLIFICATION *for* $\mathcal{P}_\forall$(S-Lin) *is* PSPACE*-complete.*

(11) SIMPLIFICATION *for* $\mathcal{P}_\forall$(Det-S-Lin) *is in* PTIME.

(12) $\mathcal{P}_\forall(\text{S-Lin}) \overset{\text{poly}}{\nrightarrow} \text{DTD}$ *and* $\mathcal{P}_\forall(\text{Det-S-Lin}) \overset{\text{poly}}{\nrightarrow} \text{DTD}$.

Proof. (1) We first show $\mathcal{P}_\exists(\text{S-Lin}) \overset{\text{poly}}{\to} \mathcal{P}_\forall(\text{S-Lin})$. The key of this proof lies in the following lemma:

Lemma 21. *For each finite set R of disjoint strongly linear expressions, a finite set S of disjoint strongly linear regular expressions can be constructed in* PTIME *such that* $\bigcup_{s\in S} L(s) = \Sigma^* \setminus \bigcup_{r\in R} L(r)$.

Before we prove this lemma, we show how it implies the theorem. For $P = \{(r_1, s_1), \ldots, (r_n, s_n)\}$, let S be the set of strongly linear expressions for $R = \{r_1, \ldots, r_n\}$ satisfying the conditions of Lemma 21. Set $P' = P \cup \bigcup_{s\in S}\{(s, \emptyset)\}$. Here, $\mathcal{T}_\exists(P) = \mathcal{T}_\exists(P')$ and since P' is disjoint and complete it follows from Lemma 12 that $\mathcal{T}_\exists(P') = \mathcal{T}_\forall(P')$. This gives us $\mathcal{T}_\exists(P) = \mathcal{T}_\forall(P')$. By Lemma 21, the set S is polynomial time computable and therefore, P' is too.

Further, note that the regular expressions in P' are copies of these in P. Therefore, $\mathcal{P}_\forall(\text{Det-S-Lin}) \overset{\text{poly}}{\to} \mathcal{P}_\exists(\text{Det-S-Lin})$ also holds. We finally give the proof of Lemma 21.

Proof of Lemma 21. For R a set of strongly linear regular expressions, let $\text{Suffix}(R) = \bigcup_{r\in R} \text{Suffix}(r)$. Define U as the set of strings aw, $a \in \Sigma$, $w \in \Sigma^*$, such that $w \in \text{Suffix}(R)$, and $aw \notin \text{Suffix}(R)$. Define V as $\text{Suffix}(R) \setminus \bigcup_{r\in R} L(r)$.

We claim that $S = \bigcup_{u\in U}\{\Sigma^* u\} \cup \bigcup_{v\in V}\{v\}$ is the desired set of regular expressions. For instance, for $R = \{\Sigma^* abc, \Sigma^* b, bc\}$ we have $U = \{bbc, cbc, ac, cc, a\}$ and $V = \{c\}$ which gives us $S = \{\Sigma^* bbc, \Sigma^* cbc, \Sigma^* ac, \Sigma^* cc, \Sigma^* a, c\}$.

It suffices to show that, given R: (1) S is finite and polynomial time computable; (2) the expressions in S are pairwise disjoint; (3) $\bigcup_{r\in R} L(r) \cap \bigcup_{s\in S} L(s) = \emptyset$; and (4) $\bigcup_{r\in R\cup S} L(r) = \Sigma^*$.

We first show (1). Every $r \in R$ is of the form w or $\Sigma^* w$, for some w. Then, for r there are only $|w|$ suffixes in $L(r)$ which can match the definition of U or V. When a string w', with $|w'| > |w|$ is a suffix in $L(r)$ then, r must be of the form $\Sigma^* w$ and thus for every $a \in \Sigma$, aw is also a suffix in $L(r)$, and thus $aw \notin U$. Further, $w' \notin V$. So, the number of strings in U and V is bounded by the number of rules in R times the length of the strings w occurring in the expressions in R, times the number of alphabet symbols, which is a polynomial. Obviously, we can also compute these strings in polynomial time.

For (2), we must check that the generated expressions are all pairwise disjoint. First, every expression generated by V defines only one string, so two expressions generated by V always have an empty intersection. For an expression $\Sigma^* aw$ generated by U and an string w' in V, suppose that their intersection is non-empty and thus $w' \in L(\Sigma^* aw)$. Then, aw must be a suffix of w' and we know by definition of V that $w' \in \text{Suffix}(R)$. But then, also $aw \in \text{Suffix}(R)$ which contradicts the definition of U. Third, suppose that two expressions $\Sigma^* aw$, $\Sigma^* a'w'$ generated by U have a non-empty intersection. Then, aw must be a suffix of $a'w'$ (or the other way around, but that is perfectly symmetrical), and since $aw \neq a'w'$, aw must be a suffix of w'. But $w' \in \text{Suffix}(R)$ and thus $aw \in \text{Suffix}(R)$ must also hold, which again contradicts the definition of U.

For (3), The strings in V are explicitly defined such that their intersection with $\bigcup_{r\in R} L(r)$ is empty. For the expression generated by U, observe that they only define words which have suffixes that cannot be suffixes of any word defined by any expression in R. Therefore, $\bigcup_{r\in R} L(r) \cap \bigcup_{s\in S} L(s) = \emptyset$.

Finally, we show (4). Let $w \notin L(r)$, for any $r \in R$. We show that there exists an $s \in S$, such that $w \in L(s)$. If $w \in V$, we are done. So assume $w \notin V$. Let $w = a_1 \cdots a_k$. Now, we go from left to right through w and search for the rightmost $l \leqslant k+1$ such that $w_l = a_l \cdots a_k \in \text{Suffix}(R)$, and $w_{l-1} = a_{l-1} \cdots a_k \notin \text{Suffix}(R)$. When $l = k+1$, $w_l = \varepsilon$. Then, w is accepted by the expression $\Sigma^* a_{l-1} \cdots a_k$, which by definition must be generated by U. It is only left to show that there indeed exists such an index l for w. Thereto, note that if $l = k+1$, then it is easy to see that $w_l = \varepsilon$ is a suffix of every string accepted by every $r \in R$. Conversely, if $l = 1$ we show that $w_l = w$ cannot be a suffix of any string defined by any $r \in R$. Suppose to the contrary that $w \in \text{Suffix}(r)$, for some $r \in R$. Let r be w_r or $\Sigma^* w_r$. If w is a suffix of w_r, then w is accepted by an expression generated by V, which case we already ruled out. If w is not a suffix of w_r, then r must be of the form $\Sigma^* w_r$ and w_r must be a suffix of w. But then, $w \in L(r)$, which also contradicts our assumptions. So, we can only conclude that $w_1 \notin \text{Suffix}(R)$. So, given that $w_{k+1} \in \text{Suffix}(R)$, and $w_1 \notin \text{Suffix}(R)$, we are guaranteed to find some l, $1 < l \leqslant k+1$, such that $w_l \in \text{Suffix}(R)$, and $w_{l-1} \notin \text{Suffix}(R)$. This concludes the proof of Lemma 21. □

(7) For $P = \{(r_1, s_1), \ldots, (r_n, s_n)\}$, let $S = \{r'_1, \ldots, r'_m\}$ be the set of strongly linear expressions for $R = \{r_1, \ldots, r_n\}$ satisfying the conditions of Lemma 21. Then, define $P' = \{(r_1, s_1), \ldots, (r_n, s_n), (r'_1, \Sigma^*), \ldots, (r'_m, \Sigma^*)\}$. Here, $\mathcal{T}_\forall(P) = \mathcal{T}_\forall(P')$ and since P' is disjoint and complete it follows from Lemma 12 that $\mathcal{T}_\exists(P') = \mathcal{T}_\forall(P')$. This gives us $\mathcal{T}_\forall(P) = \mathcal{T}_\exists(P')$. By Lemma 21, the set S is polynomial time computable and therefore, P' is too.

Further, note that the regular expressions in P' are copies of these in P. Therefore, $\mathcal{P}_\exists(\text{Det-S-Lin}) \overset{\text{poly}}{\to} \mathcal{P}_\forall(\text{Det-S-Lin})$ also holds.

(2)–(3), (8)–(9) We show $\mathcal{P}_\exists(\text{S-Lin}) \stackrel{\text{poly}}{\to} \text{EDTD}^{\text{st}}$. Since deterministic strongly-linear schemas are a subset of strongly-linear schemas, since single-type EDTDs are a subset of EDTDs and since we can translate a strongly-linear schema with universal semantics into an equivalent one with existential semantics in polynomial time (Theorem 20(7)), all other results follow.

Given P, we construct an automaton-based schema $D = (A, \lambda)$ such that $L(D) = \mathcal{T}_\exists(P)$. By Lemma 7, we can then translate D into an equivalent single-type EDTD in polynomial time. Let $P = \{(r_1, s_1), \ldots, (r_n, s_n)\}$. We define D such that when A is in state q after reading w, $\lambda(q) = s_i$ iff $w \in L(r_i)$ and $\lambda(q) = \emptyset$ otherwise. The most obvious way to construct A is by constructing DFAs for the vertical expressions and combining these by a product construction. However, this would induce an exponential blow-up. Instead, we construct A in polynomial time in a manner similar to the construction used in Proposition 5.2 in [7].

First, assume that every r_i is of the form $\Sigma^* w_i$. We later extend the construction to also handle vertical expressions of the form w_i. Define $S = \{w \mid w \in \text{Prefix}(w_i),\ 1 \leqslant i \leqslant n\}$. Then, $A = (Q, q_0, \delta)$ is defined as $Q = S \cup \{q_0\}$, and for each $a \in \Sigma$,

- $\delta(q_0, a) = a$ if $a \in S$, and $\delta(q_0, a) = q_0$ otherwise; and
- for each $w \in S$, $\delta(w, a) = w'$, where w' is the longest suffix of wa in S, and $\delta(w, a) = q_0$ if no string in S is a suffix of wa.

For the definition of λ, let $\lambda(q_0) = \emptyset$, and for all $w \in S$, $\lambda(w) = s_i$ if $w \in L(r_i)$ and $\lambda(w) = \emptyset$ if $w \notin L(r_i)$ for all $i \leqslant n$. Note that since the vertical expression are disjoint, λ is well-defined.

We prove the correctness of our construction using the following lemma which can easily be proved by induction on the length of u.

Lemma 22. *For any string $u = a_1 \cdots a_k$,*

(1) *if $q_0 \Rightarrow_{A,u} q_0$, then no suffix of u is in S; and*
(2) *if $q_0 \Rightarrow_{A,u} w$, for some $w \in S$, then w is the biggest element in S which is a suffix of u;*
(3) *$q_0 \Rightarrow_{A,u} q$, with $\lambda(q) = \emptyset$, iff $u \notin L(r_i)$, for any $i \leqslant n$; and*
(4) *$q_0 \Rightarrow_{A,u} w$, $w \in S$, with $\lambda(w) = s_i$, iff $u \in L(r_i)$.*

To show that $L(D) = \mathcal{T}_\exists(P)$, it suffices to prove that for any tree t, a node $v \in P_\exists(t)$ iff $\text{ch-str}(v) \in L(\lambda(q))$ for $q \in Q$ such that $q_0 \Rightarrow_{A,\text{anc-str}(v)} q$.

First, suppose $v \in P_\exists(t)$. Then, for some $i \leqslant n$, $\text{anc-str}(v) \in L(r_i)$ and $\text{ch-str}(v) \in L(s_i)$. By Lemma 22(4), and the definition of λ, $q_0 \Rightarrow_{\text{anc-str}(v)} q$, with $\lambda(q) = s_i$. But then, $\text{ch-str}(v) \in L(\lambda(q))$.

Conversely, suppose that for q such that $q_0 \Rightarrow_{A,\text{anc-str}(v)} q$, $\text{ch-str}(v) \in L(\lambda(q))$ holds. Then, by Lemma 22(4), there is some i such that $\text{anc-str}(v) \in L(r_i)$, and by the definition of λ, $\text{ch-str}(v) \in L(s_i)$. It follows that $v \in P_\exists(t)$.

We have now shown that the construction is correct when all expressions are of the form $\Sigma^* w$. We sketch the extension to the full class of strongly linear expressions. Assume w.l.o.g. that there exists some m such that for $i \leqslant m$, $r_i = \Sigma^* w_i$ and for $i > m$, $r_i = w_i$. Define $S = \{w \mid w \in \text{Prefix}(w_i) \wedge 1 \leqslant i \leqslant m\}$ in the same manner as above, and $S' = \{w \mid w \in \text{Prefix}(w_i) \wedge m < i \leqslant n\}$. Define $A = (Q, q_0', \delta)$, with $Q = \{q_0, q_0'\} \cup S \cup S'$. Note that the elements of S and S' need not be disjoint. Therefore, we denote the states corresponding to elements of S' by primes, for instance $ab \in S'$ corresponds to the state $a'b'$. Then, for any symbol $a \in \Sigma$, $\delta(q_0', a) = a'$ if $a \in S'$; $\delta(q_0', a) = a$ if $a \notin S' \wedge a \in S$; and $\delta(q_0', a) = q_0$ otherwise. For a string $w \in S'$, $\delta(w', a) = w'a'$ if $wa \in S'$, $\delta(w', a)$ is the longest suffix of wa in S if it exists and $wa \notin S'$, and $\delta(w', a) = q_0$ otherwise. The transition function for q_0 and the states introduced by S remains the same. So, we have added a subautomaton to A which starts by checking whether $w = w_i$, for some $i > m$, much like a suffix-tree, and switches to the normal operation of the original automaton if this is not possible anymore.

Finally, the definition of λ again remains the same for q_0 and the states introduced by S. Further, $\lambda(q_0') = \emptyset$, and $\lambda(w') = r_i$ if $w \in L(r_i)$ for some i, $1 \leqslant i \leqslant n$, and $\lambda(w') = \emptyset$ otherwise. The previous lemma can be extended for this extended construction and the correctness of the construction follows thereof.

(4), (10) This follows immediately from Theorem 16(4) and (9). The upper bound carries over since every strongly linear schema is also a linear schema. For the lower bound, observe that the schema used in the proofs of Theorem 16(4) and (9) is strongly linear.

(5), (11) We give the proof for the existential semantics. By Theorem 20(7) the result carries over immediately to the universal semantics.

The algorithm proceeds in a number of steps. First, construct an automaton-based schema D_1 such that $L(D_1) = \mathcal{T}_\exists(P)$. By Theorem 20(3) this can be done in polynomial time. Furthermore, the regular expressions in D_1 are copies of the horizontal expressions in P and are therefore also one-unambiguous. Then, translate D_1 into a single-type EDTD $D_2 = (\Sigma, \Sigma', d_2, a, \mu)$, which by Lemma 7 can again be done in PTIME and also maintains the one-unambiguity of the used regular expressions. Then, we trim D_2 which can be done in polynomial time by Lemma 5(1) and also preserves the one-unambiguity of the expressions in D_2. Finally, we claim that $L(D_2) = \mathcal{T}_\exists(P)$ is definable by a DTD iff for every two types $\mathsf{a}^i, \mathsf{a}^j \in \Sigma'$ it holds that $L(\mu(d(\mathsf{a}^i))) = L(\mu(d(\mathsf{a}^j)))$. Since all regular expressions in D_2 are one-unambiguous, this can be tested in polynomial time. We finally prove the above claim:

First, suppose that for every pair of types $\mathsf{a}^i, \mathsf{a}^j \in \Sigma'$ it holds that $\mu(d_2(\mathsf{a}^i)) = \mu(d_2(\mathsf{a}^j))$. Then, consider the DTD $D = (\Sigma, d, s)$, where $d(a) = \mu(d_2(\mathsf{a}^i))$ for some $\mathsf{a}^i \in \Sigma'$. Since all regular expression $\mu(d_2(\mathsf{a}^i))$, with $\mu(\mathsf{a}^i) = a$, are equivalent, it does not matter which type we choose. Now, $L(D) = L(D_2)$ which shows that $L(D_2)$ is definable by a DTD.

Conversely, suppose that there exist types $\mathsf{a}^i, \mathsf{a}^j \in \Sigma'$ such that $\mu(L(d(\mathsf{a}^i))) \neq \mu(L(d(\mathsf{a}^j)))$. We show that $L(D_2)$ is not closed under ancestor-guarded subtree exchange. From Lemma 4 it then follows that $L(D_2)$ is not definable by a DTD. Since $\mu(L(d(\mathsf{a}^i))) \neq \mu(L(d(\mathsf{a}^j)))$, there exists a string w such that $w \in \mu(L(d(\mathsf{a}^i)))$ and $w \notin \mu(L(d(\mathsf{a}^j)))$ or $w \notin \mu(L(d(\mathsf{a}^i)))$ and $w \in \mu(L(d(\mathsf{a}^j)))$. We consider the first case, the second is identical. Let $t_1 \in L(d_2)$ be a tree with some node v with $\mathrm{lab}^{t_1}(v) = \mathsf{a}^i$ and $\text{ch-str}^{t_1}(v) = w'$ where $\mu(w') = w$. Further, let $t_2 \in L(d_2)$ be a tree with some node u with $\mathrm{lab}^{t_2}(u) = \mathsf{a}^j$. Since D_2 is trimmed, t_1 and t_2 must exist by Lemma 5(2). Now, define $t_3 = \mu(t_2)[u \leftarrow \mu(\text{subtree}^{t_1}(v))]$ which is obtained from $\mu(t_1)$ and $\mu(t_2)$ by label-guarded subtree exchange. Because D_2 is a single-type EDTD, it must assign the type a^j to node u in t_3. However, $\text{ch-str}^{t_3}(u) = w \notin \mu(L(d(\mathsf{a}^j)))$ and thus $t_3 \notin L(D_3)$. This shows that D_2 is not closed under label-guarded subtree exchange.

(6), (12) We first show that $\mathcal{P}_\forall(\text{Det-S-Lin}) \nrightarrow \text{DTD}$ and then $\mathcal{P}_\exists(\text{S-Lin}) \overset{\text{poly}}{\nrightarrow} \text{DTD}$. Since deterministic strongly-linear schemas are a subset of strongly-linear schemas and since we can translate a strongly-linear schema with universal semantics into an equivalent one with existential semantics in polynomial time (Theorem 20(7)), all other results follow.

First, to show that $\mathcal{P}_\forall(\text{Det-S-Lin}) \nrightarrow \text{DTD}$, let $\Sigma^P = \{a, b, c, d, e, f\}$ and $P = \{(a, b + c), (ab, d), (ac, d), (abd, \varepsilon), (acd, f), (acdf, \varepsilon)\}$. Here, $a(b(d)) \in \mathcal{T}_\forall(P)$ and $a(c(d(f))) \in \mathcal{T}_\forall(P)$ but $a(b(d(f))) \notin \mathcal{T}_\forall(P)$. Therefore, $\mathcal{T}_\forall(P)$ is not closed under ancestor-guarded subtree exchange and by Lemma 4 is not definable by a DTD.

To show that $\mathcal{P}_\exists(\text{S-Lin}) \overset{\text{poly}}{\nrightarrow} \text{DTD}$, note that the algorithm in the above proof also works when the horizontal regular expressions are not one-unambiguous. The total algorithm then becomes PSPACE, because we have to test equivalence of regular expressions. However, the DTD D is still constructed in polynomial time, which completes this proof. □

6. Conclusion

In this paper, we studied the succinctness of pattern-based schemas under existential and universal semantics with respect to each other and the common schema formalisms: DTDs, EDTDs, and single-type EDTDs. This is done for regular, linear, and strongly linear pattern-based schemas. The main observation is that schemas under existential semantics behave at least as good or better than the corresponding schemas under universal semantics. In some translations a double exponential blow-up can even not be avoided. However, almost all problems for the class of strongly linear schemas turn out to be tractable, which makes this class very interesting from a practical point of view.

As our main motivation comes from using pattern-based schemas as a front-end to more traditional schema languages like XSDs, we only studied the translation of pattern-based schemas to these formalisms. However, it would also be interesting to see results for translations in the other direction. We leave open the exact complexity of translating from regular and linear schemas under existential semantics to DTDs, and of the transformation of linear schemas between the two semantics.

References

[1] A. Brüggemann-Klein, M. Murata, D. Wood, Regular tree and regular hedge languages over unranked alphabets: Version 1, April 3, 2001, Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.
[2] C. Sperberg-McQueen, H. Thompson, XML Schema, http://www.w3.org/XML/Schema, 2005.
[3] J. Clark, M. Murata, RELAX NG Specification, OASIS (December 2001). URL http://www.oasis-open.org/committees/relax-ng/spec-20011203.html.
[4] Y. Papakonstantinou, V. Vianu, DTD inference for views of XML data, in: Proc. 19th Symposium on Principles of Database Systems, PODS 2000, ACM Press, 2000, pp. 35–46.
[5] M. Murata, D. Lee, M. Mani, K. Kawaguchi, Taxonomy of XML schema languages using formal language theory, ACM Trans. Internet Technol. 5 (4) (2005) 660–704.
[6] W. Martens, F. Neven, T. Schwentick, G. Bex, Expressiveness and complexity of XML schema, ACM Trans. Database Syst. 31 (3) (2006) 770–813.
[7] G. Kasneci, T. Schwentick, The complexity of reasoning about pattern-based XML schemas, in: Proc. of the 26th ACM Symposium on Principles of Database Systems, PODS 2007, ACM Press, 2007, pp. 155–163.
[8] I. Glaister, J. Shallit, A lower bound technique for the size of nondeterministic finite automata, Inform. Process. Lett. 59 (2) (1996) 75–77.
[9] W. Gelade, F. Neven, Succinctness of the complement and intersection of regular expressions, in: Proc. 25th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2008, IBFI, 2008, pp. 325–336.
[10] A. Bruggemann-Klein, Regular expressions into finite automata, Theoret. Comput. Sci. 120 (2) (1993) 197–213.
[11] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time: Preliminary report, in: Conference Record of the 5th Annual ACM Symposium on Theory of Computing, STOC 1973, ACM Press, 1973, pp. 1–9.
[12] A. Bruggemann-Klein, D. Wood, One-unambiguous regular languages, Inform. and Comput. 142 (2) (1998) 182–206.