Optimizing Schema Languages for XML: Numerical Constraints and Interleaving

Wouter Gelade*, Wim Martens, and Frank Neven

Hasselt University and Transnational University of Limburg
School for Information Technology
{firstname.lastname}@uhasselt.be

Abstract. The presence of a schema offers many advantages in processing, translating, querying, and storage of XML data. Basic decision problems like equivalence, inclusion, and non-emptiness of intersection of schemas form the basic building blocks for schema optimization and integration, and algorithms for static analysis of transformations. It is thereby paramount to establish the exact complexity of these problems. Most common schema languages for XML can be adequately modeled by some kind of grammar with regular expressions at right-hand sides. In this paper, we observe that apart from the usual regular operators of union, concatenation and Kleene-star, schema languages also allow numerical occurrence constraints and interleaving operators. Although the expressiveness of these operators remain within the regular languages, their presence or absence has significant impact on the complexity of the basic decision problems. We present a complete overview of the complexity of the basic decision problems for DTDs, XSDs and Relax NG with regular expressions incorporating numerical occurrence constraints and interleaving. We also discuss chain regular expressions and the complexity of the schema simplification problem incorporating the new operators.

1 Introduction

XML is the lingua franca for data exchange on the Internet [1]. Within applications or communities, XML data is usually not arbitrary but adheres to some structure imposed by a schema. The presence of such a schema not only provides users with a global view on the anatomy of the data, but far more importantly, it enables automation and optimization of standard tasks like (*i*) searching, integration, and processing of XML data (cf., e.g., [11, 20, 23, 40]); and, (*ii*) static analysis of transformations (cf., e.g., [2, 15, 24, 30]). Decision problems like equivalence, inclusion and non-emptiness of intersection of schemas, hereafter referred to as *the basic decision problems*, constitute essential building blocks in solutions for the just mentioned optimization and static analysis problems. Additionally, the basic decision problems are fundamental for schema minimization (cf., e.g.,

* Research Assistant of the Fund for Scientific Research - Flanders (Belgium)

shop $\rightarrow$ regular* & discount-box*
regular $\rightarrow$ cd
discount-box $\rightarrow$ cd$^{[10,12]}$ price
cd $\rightarrow$ artist & title & price

Fig. 1. A sample schema using the numerical occurrence and interleave operators. The schema defines a shop that sells CDs and offers a special price for boxes of 10–12 CDs.

[9, 27]). Because of their widespread applicability, it is therefore important to establish the exact complexity of the basic decision problems for the various XML schema languages.

The most common schema languages for XML are DTD, XML Schema [36], and Relax NG [8] and can be modeled by grammar formalisms [29]. In particular, DTDs correspond to context-free grammars with regular expressions (REs) at right-hand sides, while Relax NG is abstracted by extended DTDs (EDTDs) [31] or equivalently, unranked tree automata [6], defining the regular unranked tree languages. While XML Schema is usually abstracted by unranked tree automata as well, recent results indicate that XSDs correspond to a strict subclass of the regular tree languages and are much closer to DTDs than to tree automata [26]. In fact, they can be abstracted by single-type EDTDs. As detailed in [25], the relationship between schema formalisms and grammars provides direct upper and lower bounds for the complexity of the basic decision problems.

A closer inspection of the various schema specifications reveals that the above abstractions in terms of grammars with regular expressions is too coarse. Indeed, in addition to the conventional regular expression operators like concatenation, union, and Kleene-star, the XML Schema and the Relax NG specification allow two other operators as well:

(1) Both the XML Schema and the Relax NG specification allow a certain form of unordered concatenation: the `ALL` and the `interleave` operator, respectively. This operator is actually the resurrection of the &-operator from SGML DTDs that was excluded from the definition of XML DTDs. Although there are restrictions on the use of `ALL` and `interleave`, we consider the operator in its unrestricted form. We refer by RE(&) to such regular expressions with the unordered concatenation operator.
(2) The XML Schema specification allows to express numerical occurrence constraints which define the minimal and maximal number of times a regular construct can be repeated. We refer by RE(#) to such regular expressions with numerical occurrence constraints.

We illustrate these additional operators in Figure 1. The formal definition is given in Section 2. Although the new operators can be expressed by the conventional regular operators, they cannot do so succinctly, which has severe implications on the complexity of the basic decision problems.

The goal of this paper is to study the complexity of the basic decision problems for DTDs, XSDs, and Relax NG with regular expressions extended with

interleaving and numerical occurrence constraints. The latter class of regular expressions is denoted by RE(#, &). As observed in Section 5, the complexity of inclusion and equivalence of RE(#, &)-expressions (and subclasses thereof) carries over to DTDs and single-type EDTDs. We therefore first establish the complexity of the basic decision problems for RE(#, &)-expressions and frequently occurring subclasses. These results are summarized in Table 1 and Table 2. Of independent interest, we introduce NFA(#, &)s, an extension of NFAs with counter and split/merge states for dealing with numerical occurrence constraints and interleaving operators. Finally, we revisit the simplification problem introduced in [26] for schemas with RE(#, &)-expressions. That is, given an extended DTD, can it be rewritten into an equivalent DTD or a single-type EDTD?

In this paper, we do not consider deterministic or one-unambiguous regular expressions which form a strict subclass of the regular expressions [7]. The reason is two-fold. First of all, one-unambiguity is a highly debatable constraint (cf., e.g., pg 98 of [38] and [22, 35]) which is only required for DTDs and XML Schema, not for Relax NG. Actually, the only direct advantage of one-unambiguity is that it gives rise to PTIME algorithms for some of the basic decision problems for standard regular expressions. The latter does not hold anymore for RE(#, &)-expressions rendering the notion even less attractive. Indeed, already intersection for one-unambiguous regular expressions is PSPACE-hard [25] and inclusion for one-unambiguous RE(#)-expressions is coNP-hard [17]. A second reason is that, in contrast to conventional regular expressions, one-unambiguity is not yet fully understood for regular expressions with numerical occurrence constraints and interleaving operators. Some initial results are provided by Bruggemann-Klein, and Kilpeläinen and Tuhkanen who give algorithms for deciding one-unambiguity of RE(&)- and RE(#)-expressions, respectively [5, 18]. No study investigating their properties has been undertaken. Such a study, although definitely relevant, is outside the scope of this paper.

Outline. In Section 2, we provide the necessary definitions. In Section 3, we define NFA(#, &). In Section 4 and Section 5, we establish the complexity of the basic decision problems for regular expressions and schema languages, respectively. We discuss simplification in Section 6. We conclude in Section 7. A version of this paper containing all proofs is available from the authors' webpages.

2 Definitions

2.1 Regular Expressions with Counting and Interleaving

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^*. A

	INCLUSION	EQUIVALENCE	INTERSECTION
RE	PSPACE ([37])	PSPACE ([37])	PSPACE ([21])
RE(&)	EXPSPACE ([28])	EXPSPACE ([28])	**PSPACE**
RE(#) and RE(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
NFA(#), NFA(&), and NFA(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
DTDs with RE	PSPACE ([37])	PSPACE ([37])	PSPACE ([21])
DTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
single-type EDTDs with RE	PSPACE ([25])	PSPACE ([25])	EXPTIME ([25])
single-type EDTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**EXPTIME**
EDTD with RE	EXPTIME ([34])	EXPTIME ([34])	EXPTIME ([33])
EDTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**EXPTIME**

Table 1. Overview of new and known complexity results. All results are completeness results. The new results are printed in bold.

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. By $w_1 \& w_2$ we denote the set of strings that is obtained by *interleaving* or *shuffling* w_1 and w_2 in every possible way. That is, for $w \in \Sigma^*$, $w \& \varepsilon = \varepsilon \& w = \{w\}$, and $a \cdot w_1 \& b \cdot w_2 = (\{a\} \cdot (w_1 \& b \cdot w_2)) \cup (\{b\} \cdot (a \cdot w_1 \& w_2))$. The operator & is then extended to languages in the canonical way.

The set of *regular expressions* over Σ, denoted by RE, is defined in the usual way: ε, and every Σ-symbol is a regular expression; and when r and s are regular expressions, then rs, $r + s$, and r^* are also regular expressions. By RE(#, &) we denote RE extended with two new operators: *interleaving* and *numerical occurrence constraints*. That is, when r and s are RE(#, &)-expressions then so are $r \,\&\, s$ and $r^{[k,\ell]}$ for $k, \ell \in \mathbb{N}$ with $k \leq \ell$ and $\ell > 0$. By RE(#) and RE(&), we denote RE extended only with counting and interleaving, respectively.

The language defined by a regular expression r, denoted by $L(r)$, is inductively defined as follows: $L(\varepsilon) = \{\varepsilon\}$; $L(a) = \{a\}$; $L(rs) = L(r) \cdot L(s)$; $L(r + s) = L(r) \cup L(s)$; $L(r^*) = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} L(r)^i$, $L(r^{[k,\ell]}) = \bigcup_{i=k}^{\ell} L(r)^i$; and, $L(r \,\&\, s) = L(r) \,\&\, L(s)$. The *size* of a regular expression r over Σ, denoted by $|r|$, is the number of Σ-symbols and operators occurring in r plus the sizes of the binary representations of the integers. By $r?$ and r^+, we abbreviate the expression $r + \varepsilon$ and rr^*, respectively. We assume familiarity with finite automata such as nondeterministic finite automata (NFAs) and deterministic finite automata (DFAs) [14].

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 $\mathrm{Dom}(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{iu \mid u \in \mathrm{Dom}(t_i)\}$. In the sequel, whenever we say tree, we always mean Σ-tree. A *tree language* is a set of trees.

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

Definition 1. Let $\mathcal{M}$ be a class of representations of regular string languages over Σ.

1. A *DTD($\mathcal{M}$)* over Σ is a tuple (Σ, d, s_d) where d is a function that maps Σ-symbols to elements of $\mathcal{M}$ and $s_d \in \Sigma$ is the start symbol. For convenience of notation, we denote (Σ, d, s_d) by d and leave the start symbol s_d implicit whenever this cannot give rise to confusion.
   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($\mathcal{M}$)) 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($\mathcal{M}$) 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 $\Sigma' = \{a^i \mid 1 \leq i \leq k_a, a \in \Sigma, i \in \mathbb{N}\}$ for some natural numbers k_a, and we set $\mu(a^i) = a$.
3. A *single-type EDTD* ($\mathrm{EDTD}^{\mathrm{st}}(\mathcal{M})$) over Σ is an EDTD($\mathcal{M}$) $D = (\Sigma, \Sigma', d, s, \mu)$ 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, EDTD(#), EDTD(&), and EDTD(#,&), the classes EDTD(RE), EDTD(RE(#)), EDTD(RE(&)), and EDTD(RE(#,&)), respectively. The same notation is used for $\mathrm{EDTD}^{\mathrm{st}}$ and DTDs.

For clarity, we write $a \to r$ rather than $d(a) = r$ in examples and proofs. Following this notation, a simple example of an EDTD is the following:

$$\begin{array}{l|l} \mathrm{shop}^1 \to (\mathrm{dvd}^1 + \mathrm{dvd}^2)^* \mathrm{dvd}^2 (\mathrm{dvd}^1 + \mathrm{dvd}^2)^* & \mathrm{title}^1 \to \varepsilon \\ \mathrm{dvd}^1 \to \mathrm{title}^1\ \mathrm{price}^1 & \mathrm{price}^1 \to \varepsilon \\ \mathrm{dvd}^2 \to \mathrm{title}^1\ \mathrm{price}^1\ \mathrm{discount}^1 & \mathrm{discount}^1 \to \varepsilon \end{array}$$

Here, dvd^1 defines ordinary DVDs, while dvd^2 defines DVDs on sale. The rule for shop^1 specifies that there should be at least one DVD on sale. Note that the above is not a single-type EDTD as dvd^1 and dvd^2 occur in the same rule.

As explained in [29, 26], EDTDs and single-type EDTDs correspond to Relax NG and XML Schema, respectively.

2.3 Decision Problems

The following problems are fundamental to this paper.

Definition 2. Let $\mathcal{M}$ be a class of regular expressions, string automata, or extended DTDs. We define the following problems:

- INCLUSION for $\mathcal{M}$: Given two elements $e, e' \in \mathcal{M}$, is $L(e) \subseteq L(e')$?
- EQUIVALENCE for $\mathcal{M}$: Given two elements $e, e' \in \mathcal{M}$, is $L(e) = L(e')$?.
- INTERSECTION for $\mathcal{M}$: Given an arbitrary number of elements $e_1, \ldots, e_n \in \mathcal{M}$, is $\bigcap_{i=1}^{n} L(e_i) \neq \emptyset$?
- MEMBERSHIP for $\mathcal{M}$: Given an element $e \in \mathcal{M}$ and a string or a tree f, is $f \in L(e)$?

We recall the known results concerning the complexity of REs and EDTDs.

Theorem 3. *(1)* INCLUSION, EQUIVALENCE, *and* INTERSECTION *for REs are* PSPACE*-complete [21, 37].*
(2) INCLUSION *and* EQUIVALENCE *for RE(&) are* EXPSPACE*-complete [28].*
(3) INCLUSION *and* EQUIVALENCE *for EDTDst are* PSPACE*-complete [25];* INTERSECTION *for EDTDst is* EXPTIME*-complete [25].*
(4) INCLUSION, EQUIVALENCE, *and* INTERSECTION *for EDTDs are* EXPTIME*-complete [33, 34].*
(5) MEMBERSHIP *for RE(&) is* NP*-complete [28].*

3 Automata for Occurrence Constraints and Interleaving

We introduce the automaton model NFA(#, &). In brief, an NFA(#, &) is an NFA with two additional features: *(i)* split and merge transitions to handle interleaving; and, *(ii)* counting states and transitions to deal with numerical occurrence constraints. The idea of split and merge transitions stems from Jędrzejowicz and Szepietowski [16]. Their automata are more general as they can express shuffle-closure which is not regular. Counting states are also used in the counter automata of Kilpeläinen and Tuhkanen [19], and Reuter [32] although these counter automata operate quite differently from NFA(#)s. Zilio and Lugiez [10] also proposed an automaton model that incorporates counting and interleaving by means of Presburger formulas. None of the cited papers consider the complexity of the basic decision problems of their model. We will use NFA(#, &)s for obtaining complexity upper bounds in Sections 4 and 5.

For readability, we denote $\Sigma \cup \{\varepsilon\}$ by Σ_ε. We then define an NFA(#, &) as follows.

Definition 4. An NFA(#, &) is a 5-tuple $A = (Q, \Sigma, s, f, \delta)$ where

- Q is a finite set of states. To every $q \in Q$, we associate a lower bound $\min(q) \in \mathbb{N}$ and an upper bound $\max(q) \in \mathbb{N}$.
- $s, f \in Q$ is the start and final state, respectively.

– δ is the transition relation and is a subset of the union of the following sets:

(1)	$Q \times \Sigma_\varepsilon \times Q$	ordinary transition (resets the counter)
(2)	$Q \times \{\text{store}\} \times Q$	transition that does not reset the counter
(3)	$Q \times \{\text{split}\} \times Q \times Q$	split transition
(4)	$Q \times Q \times \{\text{merge}\} \times Q$	merge transition

Let $\max(A) = \max\{\max(q) \mid q \in Q\}$ be the largest upper bound occurring in A. A *configuration* γ is a pair (P, α) where, $P \subseteq Q$ is a set of states and $\alpha : Q \to \{0, \ldots, \max(A)\}$ is the value function mapping states to the value of their counter. For a state $q \in Q$, we denote by α_q the value function mapping q to 1 and every other state to 0. The initial configuration γ_s is $(\{s\}, \alpha_s)$. The final configuration γ_f is $(\{f\}, \alpha_f)$. When α is a value function then $\alpha[q = 0]$ and $\alpha[q^{++}]$ denote the functions obtained from α by setting the value of q to 0 and incrementing the value of q by 1, respectively, while leaving all other values unchanged.

We now define the transition relation between configurations. Intuitively, the value of the state at which the automaton arrives is always incremented by one. When exiting a state, the state's counter is always reset to zero, except when we exit through a *counting transition*, in which case the counter remains the same. In addition, exiting a state through a non-counting transition is only allowed when the value of the counter lies between the allowed minimum and maximum. The latter, hence, ensures that the occurrence constraints are satisfied. *Split* and *merge transitions* start and close a parallel composition.

A configuration $\gamma' = (P', \alpha')$ *immediately follows* a configuration $\gamma = (P, \alpha)$ by reading $\sigma \in \Sigma_\varepsilon$, denoted $\gamma \to_{A,\sigma} \gamma'$, when one of the following conditions hold:

1. **(ordinary transition)** there is a $q \in P$ and $(q, \sigma, q') \in \delta$ such that $\min(q) \leq \alpha(q) \leq \max(q)$, $P' = (P - \{q\}) \cup \{q'\}$, and $\alpha' = \alpha[q = 0][q'^{++}]$. That is, A is in state q and moves to state q' by reading σ (note that σ can be ε). The latter is only allowed when the counter value of q is between the lower and upper bound. The state q is replaced in P by q'. The counter of q is reset to zero and the counter of q' is incremented by one.
2. **(counting transition)** there is a $q \in P$ and $(q, \text{store}, q') \in \delta$ such that $\alpha(q) < \max(q)$, $P' = (P - \{q\}) \cup \{q'\}$, and $\alpha' = \alpha[q'^{++}]$. That is, A is in state q and moves to state q' by reading ε when the counter of q has not reached its maximal value yet. The state q is replaced in P by q'. The counter of q is not reset but remains the same. The counter of q' is incremented by one.
3. **(split transition)** there is a $q \in P$ and $(q, \text{split}, q'_1, q'_2) \in \delta$ such that $\min(q) \leq \alpha(q) \leq \max(q)$, $P' = (P - \{q\}) \cup \{q'_1, q'_2\}$, and $\alpha' = \alpha[q = 0][q'^{++}_1][q'^{++}_2]$. That is, A is in state q and splits into states q'_1 and q'_2 by reading ε when the counter value of q is between the lower and upper bound. The state q in P is replaced by (split into) q'_1 and q'_2. The counter of q is reset to zero, and the counters of q'_1 and q'_2 are incremented by one.

4. **(merge transition)** there are $q_1, q_2 \in P$ and $(q_1, q_2, \text{merge}, q') \in \delta$ such that, for each $j = 1, 2$, $\min(q_j) \leq \alpha(q_j) \leq \max(q_j)$, $P' = (P - \{q_1, q_2\}) \cup \{q'\}$, and $\alpha' = \alpha[q_1 = 0][q_2 = 0][q'^{++}]$. That is, A is in states q_1 and q_2 and moves to state q' by reading ε when the respective counter values of q_1 and q_2 are between the lower and upper bounds. The states q_1 and q_2 in P are replaced by (merged into) q', the counters of q_1 and q_2 are reset to zero, and the counter of q' is incremented by one.

For a string w and two configurations γ, γ', we denote by $\gamma \Rightarrow_{A,w} \gamma'$ when there is a sequence of configurations $\gamma \rightarrow_{A,\sigma_1} \cdots \rightarrow_{A,\sigma_n} \gamma'$ such that $w = \sigma_1 \cdots \sigma_n$. The latter sequence is called a *run* when γ is the initial configuration γ_s. A string w is *accepted* by A iff $\gamma_s \Rightarrow_{A,w} \gamma_f$ with γ_f the final configuration. We usually denote $\Rightarrow_{A,w}$ simply by $\Rightarrow_w$ when A is clear from the context. We denote by $L(A)$ the set of strings accepted by A. The size of A, denoted by $|A|$, is $|Q| + |\delta| + \Sigma_{q \in Q} \log(\max(q))$. So, each $\max(q)$ is represented in binary.

An NFA(#) is an NFA(#, &) without split and merge transitions. An NFA(&) is an NFA(#, &) without counting transitions. An NFA is an NFA(#) without counting transitions. NFA(#, &) therefore accept all regular languages.

The next theorem shows the complexity of translating between RE(#, &) and NFA(#, &), and NFA(#, &) and NFA. In brief, the proof of part (1) is by induction on the structure of RE(#, &)-expressions. Figure 2 illustrates the inductive steps for expressions $r_1^{[k,\ell]}$ and $r_1 \& r_2$, employing counter, and split and merge states, respectively. For part (2), we define an NFA from an NFA(#, &) that keeps in its state the current configuration of the latter: i.e., a set of states and a value function.

Theorem 5. *(1) Given an RE(#, &)-expression r, an equivalent NFA(#, &) can be constructed in time polynomial in the size of r.*
(2) Given an NFA(#, &) A, an equivalent NFA can be constructed in time exponential in the size of A.

We next turn to the complexity of the basic decision problems for NFA(#, &).

Theorem 6. *(1)* EQUIVALENCE *and* INCLUSION *for NFA(#, &) is* EXPSPACE*-complete;*
(2) INTERSECTION *for NFA(#, &) is* PSPACE*-complete; and,*
(3) MEMBERSHIP *for NFA(#) is* NP*-hard,* MEMBERSHIP *for NFA(&), and NFA(#, &) is* PSPACE*-complete.*

We only provide some intuition. For part (1), membership in EXPSPACE follows directly from Theorem 5(2) and the fact that INCLUSION for NFAs is PSPACE-complete [37]. EXPSPACE-hardness follows from Theorem 5(1) and Theorem 7(3). For part (2), PSPACE-hardness follows from PSPACE-hardness of INTERSECTION for REs [21]. Membership in PSPACE is witnessed by an in parallel simulation of the given NFA(#, &)s on a guessed string. Finally, NP-hardness of MEMBERSHIP for NFA(#)s is by a reduction from INTEGER KNAPSACK, PSPACE-hardness of MEMBERSHIP for NFA(&)s is by a reduction from CORRIDOR TILING.

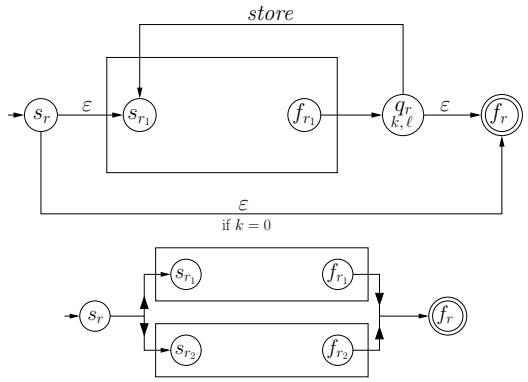

Fig. 2. From RE(#, &) to NFA(#, &).

4 Complexity of Regular Expressions

Before we turn to schemas, we first deal with the complexity of regular expressions and frequently used subclasses.

Mayer and Stockmeyer already established the EXPSPACE-completeness of INCLUSION and EQUIVALENCE for RE(&) [28]. From Theorem 5(1) and Theorem 6(1) it then directly follows that adding numerical occurrence constraints does not increase the complexity. It further follows from Theorem 5(1) and Theorem 6(2), that INTERSECTION for RE(#, &) is in PSPACE. We stress that the latter results could also have been obtained without making use of NFA(#, &) but by translating RE(#, &)s directly to NFAs. However, in the case of INTERSECTION such a construction should be done in an on-the-fly fashion in order not to go beyond PSPACE. Although such an approach is possible, we prefer the shorter and more elegant construction using NFA(#, &)s. Finally, we show that INCLUSION and EQUIVALENCE of RE(#) is also EXPSPACE-hard. While Mayer and Stockmeyer reduce from REs with intersection [12], we employ a reduction from EXP-CORRIDOR TILING.

Theorem 7. *1.* EQUIVALENCE *and* INCLUSION *for RE(#, &) is in* EXPSPACE*;*
2. INTERSECTION *for RE(#, &) is* PSPACE*-complete; and,*
3. EQUIVALENCE *and* INCLUSION *for RE(#) is* EXPSPACE*-hard.*

Proof. We prove (3). It suffices to show that it is EXPSPACE-hard to decide whether a given RE(#) defines Σ^*. The proof is a reduction from EXP-CORRIDOR TILING. A *tiling instance* is a tuple $T = (X, H, V, x_\perp, x_\top, n)$ where X is a finite set of tiles, $H, V \subseteq X \times X$ are the horizontal and vertical constraints, $x_\perp, x_\top \in X$, and n is a natural number in unary notation. A *correct exponential corridor tiling for* T is a mapping $\lambda : \{1, \ldots, m\} \times \{1, \ldots, 2^n\} \to X$ for some $m \in \mathbb{N}$ such that the following constraints are satisfied:

– the first tile of the first row is $x_\perp$: $\lambda(1, 1) = x_\perp$;

– the first tile of the m-th row is $x_\top$: $\lambda(m,1) = x_\top$;
– all vertical constraints are satisfied: $\forall i < m, \forall j \leq 2^n, (\lambda(i,j), \lambda(i{+}1,j)) \in V$; and,
– all horizontal constraints are satisfied: $\forall i \leq m, \forall j < 2^n, (\lambda(i,j), \lambda(i,j{+}1)) \in H$.

The EXP-CORRIDOR TILING problem asks, given a tiling instance, whether there exists a correct exponential corridor tiling. The latter problem is easily shown to be EXPSPACE-complete [39].

We proceed with the reduction from EXP-CORRIDOR TILING. Thereto, let $T = (X, H, V, x_\bot, x_\top, n)$ be a tiling instance. We construct an RE(#)-expression r which defines the set of all strings iff there is no correct tiling for T. As EXPSPACE is closed under complement, the EXPSPACE-hardness of EQUIVALENCE and INCLUSION for RE(#) follows.

Let $\Sigma = X \cup \{\triangle\}$. For a set $S = \{s_1, \ldots, s_k\} \subseteq \Sigma$, we abuse notation and abbreviate $(s_1 + \cdots + s_k)$ simply by S. We represent a candidate tiling consisting of m rows $\rho_1, \ldots, \rho_m$ by the string $\triangle\rho_1\triangle\cdots\triangle\rho_m\triangle$. Here, every two successive rows are delimited by the symbol $\triangle$. We now define r as a disjunction of RE(#)-expressions where every disjunct catches an error in the candidate tiling. Therefore, when r is equivalent to Σ^* there can be no correct tiling for T. It remains to define the disjuncts constituting r:

1. The string does not start or end with $\triangle$: $X\Sigma^* + \Sigma^* X$.
2. There are no 2^n tiles between two successive delimiters: $\Sigma^*\triangle(X^{[0,2^n-1]} + X^{[2^n+1,2^n+1]}X^*)\triangle\Sigma^*$.
3. The first tile is not $x_\bot$: $\triangle x\Sigma^*$ for every $x \neq x_\bot$.
4. The first tile of the last row is not $x_\top$: $\Sigma^*\triangle x X^*\triangle$ for every $x \neq x_\top$.
5. Horizontal constraint violation: $\Sigma^* x_1 x_2 \Sigma^*$ for every $(x_1, x_2) \notin H$.
6. Vertical constraint violation: $\Sigma^* x_1 \Sigma^{[2^n,2^n]} x_2 \Sigma^*$ for every $(x_1, x_2) \notin V$.

Clearly, a Σ-string that does not satisfy any of the disjuncts in r is a correct tiling for T. Hence, $L(r) \neq \Sigma^*$ iff there is a correct tiling for T. □

Bex et al. [4] established that the far majority of regular expressions occurring in practical DTDs and XSDs are of a very restricted form as defined next. The class of *chain regular expressions (CHAREs)* are those REs consisting of a sequence of factors $f_1 \cdots f_n$ where every factor is an expression of the form $(a_1{+}\cdots{+}a_n)$, $(a_1{+}\cdots{+}a_n)?$, $(a_1{+}\cdots{+}a_n)^+$, or, $(a_1{+}\cdots{+}a_n)^*$, where $n \geq 1$ and every a_i is an alphabet symbol. For instance, the expression $a(b+c)^*d^+(e+f)?$ is a CHARE, while $(ab+c)^*$ and $(a^*+b?)^*$ are not.[1]

We introduce some additional notation to define subclasses and extensions of CHAREs. By CHARE(#) we denote the class of CHAREs where also factors of the form $(a_1 + \cdots + a_n)^{[k,\ell]}$ are allowed. For the following fragments, we list the admissible types of factors. Here, a, $a?$, a^* denote the factors $(a_1 + \cdots + a_n)$, $(a_1 + \cdots + a_n)?$, and $(a_1 + \cdots + a_n)^+$, respectively, with $n = 1$, while $a\#$ denotes $a^{[k,\ell]}$, and $a\#^{>0}$ denotes $a^{[k,\ell]}$ with $k > 0$.

[1] We disregard here the additional restriction used in [3] that every symbol can occur only once.

	INCLUSION	EQUIVALENCE	INTERSECTION
CHARE	PSPACE [25]	in PSPACE [37]	PSPACE [25]
CHARE(#)	**EXPSPACE**	**in EXPSPACE**	**PSPACE**
CHARE$(a, a?)$	CONP [25]	in PTIME [25]	NP [25]
CHARE(a, a^*)	CONP [25]	in PTIME [25]	NP [25]
CHARE$(a, a?, a\#)$	**PSPACE-hard / in EXPSPACE**	**in PTIME**	**NP**
CHARE$(a, a\#^{>0})$	**in PTIME**	**in PTIME**	**in PTIME**

Table 2. Overview of new and known complexity results concerning Chain Regular Expressions. All results are completeness results, unless otherwise mentioned. The new results are printed in bold.

Table 2 lists the new and the relevant known results. We first show that adding numerical occurrence constraints to CHAREs increases the complexity of INCLUSION by one exponential. Again we reduce from EXP-CORRIDOR TILING.

Theorem 8. INCLUSION *for CHARE(#) is* EXPSPACE*-complete.*

Adding numerical occurrence constraints to the fragment CHARE$(a, a?)$ and CHARE(a, a^*), makes INCLUSION PSPACE-hard but keeps EQUIVALENCE in PTIME and INTERSECTION in NP.

Theorem 9. *(1)* EQUIVALENCE *for CHARE$(a, a?, a\#)$ is in* PTIME*.*
(2) INCLUSION *for CHARE$(a, a?, a\#)$ is* PSPACE*-hard and in* EXPSPACE*.*
(3) INTERSECTION *for CHARE$(a, a?, a\#)$ is* NP*-complete.*

Finally, we exhibit a tractable subclass with numerical occurrence constraints:

Theorem 10. INCLUSION*,* EQUIVALENCE*, and* INTERSECTION *for CHARE$(a, a\#^{>0})$ are in* PTIME*.*

5 Complexity of Schemas

5.1 DTDs and Single-Type EDTDs

In [25] it was shown for any subclass of the REs that the complexity of INCLUSION and EQUIVALENCE is the same as the complexity of the corresponding problem for DTDs and single-type EDTDs. We next generalize this result to RE$(\#, \&)$. As a corollary, all results of the previous section carry over to DTDs and single-type DTDs. The same holds for INTERSECTION and DTDs.

We call a complexity class $\mathcal{C}$ *closed under positive reductions* if the following holds for every $O \in \mathcal{C}$. Let L' be accepted by a deterministic polynomial-time Turing machine M with oracle O (denoted $L' = L(M^O)$). Let M further have the property that $L(M^A) \subseteq L(M^B)$ whenever $A \subseteq B$. Then L' is also in $\mathcal{C}$. For a more precise definition of this notion we refer the reader to [13]. For our purposes, it is sufficient that important complexity classes like PTIME, NP, CONP, PSPACE, and EXPSPACE have this property, and that every such class contains PTIME.

Proposition 11. *Let $\mathcal{R}$ be a subclass of RE(#, &) and let $\mathcal{C}$ be a complexity class closed under positive reductions. Then the following are equivalent:*

(a) INCLUSION *for $\mathcal{R}$ expressions is in $\mathcal{C}$.*
(b) INCLUSION *for DTD($\mathcal{R}$) is in $\mathcal{C}$.*
(c) INCLUSION *for EDTDst($\mathcal{R}$) is in $\mathcal{C}$.*

The corresponding statement holds for EQUIVALENCE.

The previous proposition can be generalized to INTERSECTION of DTDs as well. The proof carries over literally from [25].

Proposition 12. *Let $\mathcal{R}$ be a subclass of RE(#, &) and let $\mathcal{C}$ be a complexity class which is closed under positive reductions. Then the following are equivalent:*

(a) INTERSECTION *for $\mathcal{R}$ expressions is in $\mathcal{C}$.*
(b) INTERSECTION *for DTD($\mathcal{R}$) is in $\mathcal{C}$.*

The above proposition does not hold for single-type EDTDs. Indeed, there is a class of regular expressions $\mathcal{R}'$ for which INTERSECTION is NP-complete while INTERSECTION for EDTD$^{\text{st}}(\mathcal{R}')$ is EXPTIME-complete [25].

5.2 Extended EDTDs

We next consider the complexity of the basic decision problems for EDTDs with numerical occurrence constraints and interleaving. As the basic decision problems are EXPTIME-complete for EDTD(RE), the straightforward approach of translating every RE(#, &)-expression into an NFA and then applying the standard algorithms gives rise to a double exponential time complexity. By using NFA(#, &), we can do better: EXPSPACE for INCLUSION and EQUIVALENCE, and, more surprisingly, EXPTIME for INTERSECTION.

Theorem 13. *(1)* EQUIVALENCE *and* INCLUSION *for EDTD(#,&) is in* EXPSPACE*;*
(2) EQUIVALENCE *and* INCLUSION *for EDTD(#) and EDTD(&) is* EXPSPACE*-hard; and,*
(3) INTERSECTION *for EDTD(#,&) is* EXPTIME*-complete.*

Proof (Sketch).
(1) Given two EDTDs $D_1 = (\Sigma, \Sigma_1', d_1, s_1, \mu_1)$ and $D_2 = (\Sigma, \Sigma_2', d_2, s_2, \mu_2)$, we compute a set E of pairs $(C_1, C_2) \in 2^{\Sigma_1'} \times 2^{\Sigma_2'}$ where $(C_1, C_2) \in E$ iff there exists a tree t such that $C_j = \{\tau \in \Sigma_j' \mid t \in L((D_j, \tau))\}$ for each $j = 1, 2$. Here, (D_j, τ) denotes the EDTD D_j with start symbol τ. So, every C_j is the set of types that can be assigned by D_j to the root of t. Or when viewing D_j as a tree automaton, C_j is the set of states that can be assigned to the root in a run on t. The tree t is called a *witness tree*. Then, $t \in L(D_1)$ (resp., $t \in L(D_2)$) if $s_1 \in C_1$ (resp. $s_2 \in C_2$). Hence, $L(D_1) \not\subseteq L(D_2)$ iff there exists a pair $(C_1, C_2) \in E$ with $s_1 \in C_1$ and $s_2 \notin C_2$.

Although each witness tree can have exponential depth and therefore double exponential size, we do not need to compute it directly. Instead, we compute the set E in a bottom-up fashion where we make use of an NFA(#, &)-representation of the RE(#, &)-expressions.

(2) Is immediate from Theorem 3(2) and Theorem 7(2).

(3) In brief, given a set of EDTDs, we construct an alternating polynomial space TM which incrementally guesses a tree defined by all schemas. To be precise, the algorithm guesses the first-child-next-sibling encoding of the unranked tree. Again, RE(#, &)-expressions are translated into equivalent NFA(#, &)s.□

6 Simplification

The simplification problem is defined as follows: Given an EDTD, check whether it has an equivalent EDTD of a restricted type, i.e., an equivalent DTD or single-type EDTD. In [26], this problem was shown to be EXPTIME-complete for EDTDs with standard regular expressions. We revisit this problem in the context of RE(#, &).

Theorem 14. Given an EDTD(#,&), deciding whether it is equivalent to an EDTD$^{\text{st}}$(#,&) or DTD(#,&) is EXPSPACE-complete.

Proof (Sketch). We only show that the problem is hard for EXPSPACE. We use a reduction from universality of RE(#, &), i.e., deciding whether an RE(#, &)-expression is equivalent to Σ^*. The proof of Theorem 7(2) shows that the latter is EXPSPACE-hard. To this end, let r be an RE(#, &)-expression over Σ and let b and s be two symbols not occurring in Σ. By definition, $L(r) \neq \emptyset$. Define $D = (\Sigma \cup \{b, s\}, \Sigma \cup \{s, b^1, b^2\}, d, s, \mu)$ as the EDTD with the following rules: $s \to (b^1)^* b^2 (b^1)^*$, $b^1 \to \Sigma^*$, and $b^2 \to r$, where for every $\tau \in \Sigma \cup \{s\}$, $\mu(\tau) = \tau$, and $\mu(b^1) = \mu(b^2) = b$. We claim that D is equivalent to a single-type DTD or a DTD iff $L(r) = \Sigma^*$. Clearly, if r is equivalent to Σ^*, then D is equivalent to the DTD (and therefore also to a single-type EDTD) with rules: $s \to b^*$ and $b \to \Sigma^*$. Conversely, suppose that there exists an EDTD$^{\text{st}}$ which defines the language $L(D)$. Towards a contradiction, assume that r is not equivalent to Σ^*. Let w_r be a string in $L(r)$ and let $w_{\neg r}$ be a Σ-string not in $L(r)$. Consider the trees $t_1 = s(b(w_r)b(w_{\neg r}))$ and $t_2 = s(b(w_{\neg r})b(w_r))$. Clearly, t_1 and t_2 are in $L(D)$. However, the tree $t = s(b(w_{\neg r})b(w_{\neg r}))$ obtained from t_1 by replacing its left subtree by the left subtree of t_2 is not in $L(D)$. According to Theorem 7.1 in [26], every tree language defined by a single-type EDTD is closed under such an exchange of subtrees. So, this means that $L(D)$ cannot be defined by an EDTD$^{\text{st}}$, which leads to the desired contradiction. □

7 Conclusion

The present work gives an overview of the complexity of the basic decision problems for abstractions of several schema languages including numerical occurrence

constraints and interleaving. W.r.t. INTERSECTION the complexity remains the same, while for INCLUSION and EQUIVALENCE the complexity increases by one exponential for DTDs and single-type EDTDs, and goes from EXPTIME to EXPSPACE for EDTDs. The results w.r.t. CHAREs also follow this pattern. We further showed that the complexity of simplification increases to EXPSPACE.

We emphasize that this is a theoretical study delineating the worst case complexity boundaries for the basic decision problems. Although these complexities must be studied, we note that the regular expressions used in the hardness proofs do not correspond at all to those employed in practice. Further, w.r.t. XSDs, our abstraction is not fully adequate as we do not consider the one-unambiguity (or unique particle attribution) constraint. However, it is doubtful that this constraint is the right one to get tractable complexities for the basic decision problems. Indeed, already intersection for unambiguous regular expressions is PSPACE-hard [25] and inclusion for one-unambiguous RE(#)-expressions is coNP-hard [17]. It would therefore be desirable to find robust subclasses for which the basic decision problems are in PTIME.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.
2. M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *PODS 2005*, pages 25–36, 2005.
3. G.J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *VLDB 2006*, pages 115–126, 2006.
4. G.J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML schema: A practical study. In *WebDB 2004*, pages 79–84, 2004.
5. A. Brüggemann-Klein. Unambiguity of extended regular expressions in SGML document grammars. In *ESA 1993*, pages 73–84, 1993.
6. A. Brüggemann-Klein, M. Murata, and 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.
7. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.
8. J. Clark and M. Murata. *RELAX NG Specification*. OASIS, December 2001.
9. J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In *FCT 2005*, pages 68–79. Springer, 2005.
10. S. Dal-Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In *RTA*, pages 246–263, 2003.
11. Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing Semistructured Data with STORED. In *SIGMOD 1999*, pages 431–442, 1999.
12. M. Fürer. The complexity of the inequivalence problem for regular expressions with intersection. In *ICALP 1980*, pages 234–245. Springer, 1980.
13. L. Hemaspaandra and M. Ogihara. *Complexity Theory Companion*. Springer, 2002.
14. J.E. Hopcroft, R. Motwani, and J.D. Ullman and. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, second edition, 2001.
15. H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Trans. Inter. Tech.*, 3(2):117–148, 2003.

16. J. Jędrzejowicz and A. Szepietowski. Shuffle languages are in P. *Theoretical Computer Science*, 250(1-2):31–53, 2001.
17. P. Kilpeläinen. Inclusion of unambiguous #REs is NP-hard. Unpublished note, University of Kuopio, Finland, May 2004.
18. P. Kilpeläinen and R. Tuhkanen. One-unambiguity of regular expressions with numeric occurrence indicators. Tech. Rep. A/2006/2, Univ. Kuopio, Finland, 2006.
19. P. Kilpeläinen and R. Tuhkanen. Towards efficient implementation of XML schema content models. In *DOCENG 2004*, pages 239–241. ACM, 2004.
20. C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. Schema-based scheduling of event processors and buffer minimization for queries on structured data streams. In *VLDB 2004*, pages 228–239, 2004.
21. D. Kozen. Lower bounds for natural proof systems. In *FOCS 1977*, pages 254–266. IEEE, 1977.
22. M. Mani. Keeping chess alive — Do we need 1-unambiguous content models? In *Extreme Markup Languages*, Montreal, Canada, 2001.
23. I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries on Heterogeneous Data Sources. In *VLDB 2001*, pages 241–250, 2001.
24. W. Martens and F. Neven. Frontiers of tractability for typechecking simple XML transformations. *Journal of Computer and System Sciences*, 2006. To Appear.
25. W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for simple regular expressions. In *MFCS 2004*, pages 889–900, Berlin, 2004. Springer.
26. W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML schema. *ACM Trans. Database Systems*, 31(3), 2006. To appear.
27. W. Martens and J. Niehren. Minimizing tree automata for unranked trees. In *DBPL 2005*, pages 232–246, 2005.
28. A. J. Mayer and L. J. Stockmeyer. Word problems — this time with interleaving. *Information and Computation*, 115(2):293–311, 1994.
29. M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Inter. Tech.*, 5(4):1–45, 2005.
30. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, page To appear, 2006.
31. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *PODS 2000*, pages 35–46, New York, 2000. ACM Press.
32. F. Reuter. An enhanced W3C XML Schema-based language binding for object oriented programming languages. Manuscript, 2006.
33. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.
34. H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60, 1994.
35. C.M. Sperberg-McQueen. XML Schema 1.0: A language for document grammars. In *XML 2003*, 2003.
36. C.M. Sperberg-McQueen and H. Thompson. XML Schema. http://www.w3.org/XML/Schema, 2005.
37. L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC 1973*, pages 1–9. ACM Press, 1973.
38. E. van der Vlist. *XML Schema*. O'Reilly, 2002.
39. P. van Emde Boas. The convenience of tilings. In *Complexity, Logic and Recursion Theory*, volume 187 of *Lec. Notes in Pure and App. Math.*, pages 331–363. 1997.
40. G. Wang, M. Liu, J. X. Yu, B. Sun, G. Yu, J. Lv, and H. Lu. Effective schema-based XML query optimization techniques. In *IDEAS 2003*, pages 230–235, 2003.