Complexity of Decision Problems for Simple Regular Expressions

Wim Martens[1], Frank Neven[1], and Thomas Schwentick[2]

[1] Limburgs Universitair Centrum
Universitaire Campus
B-3590 Diepenbeek, Belgium
{wim.martens,frank.neven}@luc.ac.be
[2] Philipps Universität Marburg
Fachbereich 12, Mathematik und Informatik
tick@informatik.uni-marburg.de

Abstract. We study the complexity of the inclusion, equivalence, and intersection problem for simple regular expressions arising in practical XML schemas. These basically consist of the concatenation of factors where each factor is a disjunction of strings possibly extended with '$*$' or '?'. We obtain lower and upper bounds for various fragments of simple regular expressions. Although we show that inclusion and intersection are already intractable for very weak expressions, we also identify some tractable cases. For equivalence, we only prove an initial tractability result leaving the complexity of more general cases open. The main motivation for this research comes from database theory, or more specifically XML and semi-structured data. We namely show that all lower and upper bounds for inclusion and equivalence, carry over to the corresponding decision problems for extended context-free grammars and single-type tree grammars, which are abstractions of DTDs and XML Schemas, respectively. For intersection, we show that the complexity only carries over for DTDs.

1 Introduction

XML (eXtensible Markup Language) is becoming the standard data exchange format for the Web. Within a community, parties usually agree to only produce XML data conforming to a certain format. The presence of such a schema improves the efficiency of many tasks like, for instance, query processing, query optimization, and automatic data integration. For typechecking or type inference [11,14,18,21], schema information is even crucial. As standard decision problems of schema languages, like inclusion, equivalence, and non-emptiness of intersection, are among the basic building blocks for many of the algorithms for those problems, it is critical to establish their exact complexity.

Among the various proposals for XML schema languages, DTD (Document Type Definition) and XML Schema Definitions (XSDs) [8,9] are the most widely spread. Generally these languages are abstracted by extended context-free

J. Fiala et al. (Eds.): MFCS 2004, LNCS 3153, pp. 889–900, 2004.

Table 1. Possible factors in simple regular expressions and how they are denoted ($a \in \Sigma$, $w \in \Sigma^*$).

Factor	Abbr.
a	a
a^*	a^*
$a?$	$a?$
$(a_1 + \cdots + a_n)$	$(+a)$

Factor	Abbr.
$(a_1 + \cdots + a_n)^*$	$(+a)^*$
$(a_1 + \cdots + a_n)?$	$(+a)?$
$w?$	$w?$
w^*	w^*

Factor	Abbr.
$(a_1^* + \cdots + a_n^*)$	$(+a^*)$
$(w_1 + \cdots + w_n)$	$(+w)$
$(w_1 + \cdots + w_n)?$	$(+w)?$
$(w_1 + \cdots + w_n)^*$	$(+w)^*$

grammars (ECFGs) and unranked tree automata, respectively [20,26]. The former are context-free grammars with regular expressions as right-hand sides of rules, while the latter are a natural extension of classical tree automata to trees where nodes can have an unbounded number of children [3]. A formalism equivalent to such tree automata but which is grammar based are specialized DTDs (SDTDs) [21]. The complexity of the three afore mentioned problems is known and is PSPACE-complete for DTDs (as it reduces to the corresponding problems of regular expressions) and EXPTIME-complete for tree automata or specialized DTDs [13,22,23,24].

In the present paper, we revisit the complexity of the inclusion, equivalence, and intersection problem for XML schemas occurring effectively in practice. For instance, the PSPACE-hardness of inclusion of DTDs crucially depends on the presence of involved regular expressions that are quite unlikely to occur in realistic DTDs. Actually, a study by Bex, Neven, and Van den Bussche [2] reveals that more than 90 percent of the regular expressions occurring in practical DTDs and XSDs are of the following simple form: $e_1 \cdots e_n$ where every e_i is a factor of the form $(w_1 + \cdots + w_n)$ possibly extended with Kleene-star or question mark, and each w_i is a string (cf. Section 2 for a detailed definition and Table 1 for a list of allowed factors together with their abbreviated notation). Further, Murata et al. argued that XML Schemas do not correspond to the full class of tree automata or SDTDs, but to a strict subset of those, namely, single-type SDTDs [19].

Clearly, complexity lower bounds for the inclusion, equivalence, or the intersection problem for a class of regular expressions $\mathcal{R}$ imply lower bounds for the corresponding decision problems for DTDs and (single-type) SDTDs with right-hand sides in $\mathcal{R}$. Interestingly, we show that for inclusion and equivalence, also upper bounds for the string case carry over to DTDs and single-type SDTDs. For intersection, the latter still holds for DTDs but not for single-type SDTDs. So, it suffices to restrict attention to the complexity of simple regular expressions to derive complexity bounds for XML schema languages.

Our results on the complexity of simple regular expressions are summarized in Table 2. We denote by $\mathrm{RE}(\mathcal{S})$, the set of all simple regular expressions. Recall that the three decision problems are PSPACE-complete for the class of all regular expressions [13,24]. We briefly discuss our results.

– We show that inclusion is already CONP-complete for very innocent expressions: where every factor is of the form a or a^*, or of the form a or $a?$

Table 2. Summary of Results. Theorem numbers are given in brackets.

RE-fragment	Inclusion	Equivalence	Intersection
a, a^+	in PTIME (DFA!)	in PTIME	in PTIME (9)
a, a^*	CONP-complete (4)	in PTIME (6)	NP-complete (7)
$a, a?$	CONP-complete (4)	in PTIME (6)	NP-complete (7)
$\mathcal{S} - \{(+a)^*, (+w)^*\}$	CONP-complete (4)	in CONP	NP-complete (7)
$a, (+a)^*$	PSPACE-complete (4)	in PSPACE	NP-complete (7)
$\mathcal{S} - \{(+w)^*\}$	PSPACE-complete (4)	in PSPACE	NP-complete (7)
$\mathcal{S}$	PSPACE-complete (4)	in PSPACE	in PSPACE
$\mathrm{RE}^{\leq k}$ $(k \geq 3)$	PTIME (5)	PTIME	PSPACE-complete (8)
one-unambiguous	in PTIME	in PTIME	PSPACE-complete (8)

with a an arbitrary alphabet symbol. Even worse, when factors of the form $(a_1 + \cdots + a_n)^*$ are added we already obtain the maximum complexity: PSPACE. When such factors are disallowed the complexity remains CONP. If the number of occurrences of the same symbol in expressions is bounded by some k ($\mathrm{RE}^{\leq k}$), inclusion is in PTIME. As the running time is n^k, k should of course be small to be feasible.

- The precise complexity of the equivalence problem largely remains open. Of course, it is never harder than inclusion but we conjecture that it is tractable for a large fragment of $\mathrm{RE}(\mathcal{S})$. We only prove a PTIME upper bound for expressions where each factor is a or a^*, or a or $a?$. Even for these restricted fragments the proof is non-trivial. Basically, we show that two expressions are equivalent iff they have the same sequence normal form modulo one rewrite rule. Interestingly, the sequence normal form specifies factors much in the same way as XML Schema does. For every symbol an explicit upper and lower bound is specified. For instance, $aa^*bbc?c?$ becomes $a[1, *]b[2, 2]c[0, 2]$.
- Intersection is also CONP-hard when each factor is either of the form a or a^*, or of the form a or $a?$. However, the complexity is not always the same as for inclusion. In fact, there are cases where inclusion is harder and others where intersection is harder. Indeed, intersection remains in CONP even if we allow all kinds of factors except $(w_1 + \cdots + w_n)^*$. On the other hand, the intersection problem is PSPACE-hard for $\mathrm{RE}^{\leq 3}$. The only tractable fragment we obtain is when each factor is restricted to a or a^+.

The complexities of equivalence, inclusion and intersection for general regular expressions and several fragments were studied in [12,13,24]. From these, the most related result is the CONP-completeness of equivalence and inclusion of bounded languages [12]. A language L is bounded if there are strings $v_1, \ldots, v_n$ such that $L \subseteq v_1^* \cdots v_n^*$. It should be noted that the latter is much more general than, e.g., our $\mathrm{RE}(w^*)$. More recently, two fragments of simple regular expressions have been shown to be tractable: inclusion for $\mathrm{RE}(a?, (+a)^*)$ [1], and $\mathrm{RE}(a, \Sigma, \Sigma^*)$ [17]. This last result should be contrasted with the PSPACE-completeness of inclusion for $\mathrm{RE}(a, (+a), (+a)^*)$.

We conclude by a remark on one-unambiguous or deterministic regular expressions. Basically, these are regular expressions which have a deterministic Glushkov automaton [4]. The XML specification requires DTD content models to be deterministic because of compatibility with SGML (Section 3.2.1 of [8]). Of course, for such expressions, inclusion and equivalence are in PTIME. Nevertheless, intersection remains PSPACE-hard (cf. Theorem 8). Unfortunately, the notion of deterministic content models is not a transparent one for the average user, as is witnessed by practical studies [7,2] which found a number of non-deterministic content models in actual DTDs. Actually, for this very reason Clarke and Murata abandoned the notion in their Relax NG specification [25]. Hence, from a scientific viewpoint, we believe it makes sense to study the broader class of possibly non-deterministic but simple and practical regular expressions. Another consequence of our results, independent of the one-unambiguous issue, is that optimization problems for navigational queries as expressed by caterpillar expressions [5], $\mathcal{X}_{\text{reg}}^{\text{CPath}}$ and $\mathcal{X}_{\text{reg}}$ [16], or regular path queries [6] quickly turn intractable. Due to space limitations many proofs are omitted. We refer the interested reader to [15].

2 Definitions

Regular expressions. For the rest of the paper let Σ denote a finite alphabet. A Σ-string (or simply string) is a finite sequence $w = a_1 \cdots a_n$ of Σ-symbols. We denote the length of w by $|w|$. The empty string is denoted by ε. The set of all strings is denoted by Σ^*. The syntax and semantics of regular expressions is defined in the usual way. By $L(r)$ we denote the language defined by regular expression r. By $r?$ and r^+, we abbreviate the expressions $r + \varepsilon$ and rr^*. Sometimes, we denote $w \in L(r)$ simply by $w \in r$.

We consider simple regular expressions occurring in practice in DTDs and XML Schemas [7]. The vast majority of these can be defined as follows.

Definition 1. A *base symbol* is a regular expression s, $s?$, or s^* where s is a non-empty string; a *factor* is of the form e, e^*, or $e?$ where e is a disjunction of base symbols. A *simple regular expression* is ε, $\emptyset$, or a sequence of factors.

An example of a simple regular expression is $((abc)^* + b^*)(a + b)?(ab)^*(ac + b)^*$. We introduce a uniform syntax to denote subclasses of simple regular expressions by specifying the allowed factors. We distinguish whether the string s of a base symbol consists of a single symbol (a) or a string (w) and whether it is extended by ? or $*$. Further, we distinguish between factors with one disjunct or with arbitrarily many disjuncts, the latter is denoted by $(+\cdots)$. Finally, factors can again be extended by $*$ or ?. A list of possible factors is displayed in Table 1. This table only shows types that are different. E.g., we write $\text{RE}((+a)^*, w?)$ for the set of regular expressions $e_1 \cdots e_n$ where every e_i is $(a_1 + \cdots + a_n)^*$ for some $a_1, \ldots, a_n \in \Sigma$ and $n \geq 1$, or $w?$ for some $w \in \Sigma^*$.

If $A = \{a_1, \ldots, a_n\}$ is a set of symbols, we often denote $(a_1 + \ldots + a_n)$ simply by A. We denote the class of all simple regular expressions by $\text{RE}(\mathcal{S})$.

Decision problems. Two regular expressions r, r' are *included*, denoted $r \subseteq r'$ iff $L(r) \subseteq L(r')$. They are *equivalent*, denoted $r \equiv r'$, if $L(r) = L(r')$. The following three problems are fundamental to this paper.

Definition 2. Let $\mathcal{R}$ be a class of regular expressions.

- The *inclusion problem* for $\mathcal{R}$, is to test for two given expressions $r, r' \in \mathcal{R}$, whether $r \subseteq r'$.
- The *equivalence problem* for $\mathcal{R}$, is to test for two expressions $r, r' \in \mathcal{R}$, whether $r \equiv r'$.
- The *intersection problem* for $\mathcal{R}$, is to determine for an arbitrary number of expressions $r_1, \ldots, r_n \in \mathcal{R}$, whether $\bigcap_{i=1}^{n} L(r_i) \neq \emptyset$.

In the sequel, we abuse notation and denote $\bigcap_{i=1}^{n} L(r_i) \neq \emptyset$ simply by $\bigcap_{i=1}^{n} r_i \neq \emptyset$. For arbitrary regular expressions these problems are PSPACE-complete [13,24].

3 Decision Problems for DTDs and XML Schemas

As explained in the introduction, our main motivation for this study comes from reasoning about XML schemas. In this section, we describe how the basic decision problems for such schemas, namely whether two schemas describe the same set of documents or whether one describes a subset of the other, basically reduce to the equivalence and inclusion problem for regular expressions. We also address the problem whether a set of schemas define a common XML document. W.r.t. DTDs the latter problem again reduces to the corresponding problem for regular expressions; for XML Schemas it does not. The reader not interested in XML can safely skip this section.

3.1 XML Schema Languages

It is common to view XML documents as finite trees with labels from a finite alphabet Σ. There is no limit on the number of children of a node. Of course, elements in XML documents can also contain references to nodes. But as XML schema languages usually do not constrain these nor the data values at leaves, it is safe to view schemas as simply defining tree languages over a finite alphabet. In the rest of this section, we introduce the necessary background concerning XML schema languages. For a formal definition of Σ-trees see, e.g., [20].

Definition 3. A *DTD* is a pair (d, s_d) where d is a function that maps Σ-symbols to regular expressions and $s_d \in \Sigma$ is the start symbol. We usually simply denote (d, s_d) by d. A tree t satisfies d if its root is labeled by s_d and, for every node u with label a, the sequence $a_1 \cdots a_n$ of labels of its children is in $L(d(a))$. By $L(d)$ we denote the set of trees defined by d.

A simple example of a DTD defining the inventory of a store is the following: store $\rightarrow$ dvd dvd*, dvd $\rightarrow$ title price.

For clarity, in examples we write $w \rightarrow r$ rather than $d(w) = r$. We recall the definition of a specialized DTD from [21].

Definition 4. A *specialized DTD* (SDTD) is a 4-tuple $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$, where Σ' is an alphabet of *types*, d is a DTD over Σ' and μ is a mapping from Σ' to Σ. Note that μ can be extended to define a homomorphism on trees. A tree t then *satisfies* a specialized DTD if $t = \mu(t')$ for some $t' \in L(d)$. Again, we denote by $L(\mathbf{d})$ the set of trees defined by $\mathbf{d}$.

The class of tree languages defined by SDTDs corresponds precisely to the regular (unranked) tree languages [3]. 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 set $\mu(a^i) = a$. If a node is labeled by some a^i we call a^i the *type* of the node. A simple example of an SDTD is the following:

$$\begin{array}{ll} \text{store}^1 \rightarrow (\text{dvd}^1 + \text{dvd}^2)^*\text{dvd}^2(\text{dvd}^1 + \text{dvd}^2)^*\text{dvd}^2(\text{dvd}^1 + \text{dvd}^2)^* & \\ \text{dvd}^1 \rightarrow \text{title}^1\ \text{price}^1 & \text{dvd}^2 \rightarrow \text{title}^1\ \text{price}^1\ \text{discount}^1 \end{array}$$

Here, dvd^1 defines ordinary DVDs while dvd^2 defines DVDs on sale. The rule for store specifies that there should be at least two DVDs on discount. The following restriction on SDTDs corresponds to the expressiveness of XML schema languages [19].

Definition 5. A *single-type SDTD* (SDTD^{st}) is an SDTD $(\Sigma, \Sigma', d, \mu)$ with the property that in each regular expression $d(a)$ no two types b^i and b^j with $i \neq j$ occur.

The above defined SDTD is not single type as both dvd^1 and dvd^2 occur in the rule for store. The classes of tree languages defined by the grammars introduced above are included as follows: DTD $\subsetneq$ SDTD^{st} $\subsetneq$ SDTD [3,19].

Remark 1. There is a very simple deterministic algorithm to check validity of a tree t with respect to a SDTD^{st} $\mathbf{d}$. It proceeds top-down and assigns to every node with some symbol a a type a^i. To the root the start symbol of d is assigned; then, for every interior node u with type a^i, it is checked whether the children of u match $\mu(d(a^i))$; if not the tree is rejected, otherwise, as $\mathbf{d}$ is single-type, to each child a unique type can be assigned. The tree is accepted, if this process terminates at the leaves without any rejection. □

The *inclusion problem*, *equivalence problem* and *intersection problem* for DTDs, SDTDs and SDTD^{st}s is defined analogously to the corresponding problems for regular expressions.

3.2 Inclusion and Equivalence of XML Schema Languages

As already mentioned, testing equivalence and inclusion of XML schema languages is related to testing equivalence and inclusion of regular expressions. It is immediate that complexity lower bounds for regular expressions imply lower bounds for XML schema languages. A consequence is that testing equivalence and inclusion of XML schemas is PSPACE-hard, which suggests looking for simpler regular expressions.

Interestingly, in the case of the practically important DTDs and single-type SDTDs, it turns out that the complexities of the equivalence and inclusion problem on strings also imply *upper bounds* for the corresponding problems on XML trees.

For $\mathcal{R}$ a class of regular expressions, we denote by DTD($\mathcal{R}$), SDTD($\mathcal{R}$), and SDTD$^{\text{st}}$($\mathcal{R}$), the class of DTDs, SDTDs and SDTD$^{\text{st}}$s with regular expressions in $\mathcal{R}$.

We call a complexity class $\mathcal{C}$ *closed under positive reductions* if the following holds for every $L \in \mathcal{C}$: 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 [10]. For our purposes it is sufficient that all important complexity classes like PTIME, NP, CONP, and PSPACE have this property and that every such class contains PTIME.

Theorem 1. *Let $\mathcal{R}$ be a class of regular expressions and $\mathcal{C}$ be a complexity class which is closed under positive reductions. Then the following are equivalent.*

(a) The inclusion problem for $\mathcal{R}$ expressions is in $\mathcal{C}$.
(b) The inclusion problem for DTD($\mathcal{R}$) is in $\mathcal{C}$.
(c) The inclusion problem for SDTDst($\mathcal{R}$) is in $\mathcal{C}$.

The corresponding statement holds for the equivalence problem.

3.3 Intersection of DTDs and XML Schemas

We show in this section that the complexity of the intersection problem for regular expressions is an upper bound for the corresponding problem on XML trees. The latter is not the case for single-type DTDs.

Theorem 2. *Let $\mathcal{R}$ be a class of regular expressions and let $\mathcal{C}$ be a complexity class which is closed under positive reductions. Then the following are equivalent.*

(a) The intersection problem for $\mathcal{R}$ expressions is in $\mathcal{C}$.
(b) The intersection problem for DTD($\mathcal{R}$) is in $\mathcal{C}$.

The following theorem shows that single-type SDTDs probably cannot be included in Theorem 2 as by Theorem 7, intersection of RE$((+a), w?, (+a)?, (+a)^*)$ is in NP. The reduction is similar to the proof that intersection of deterministic top-down tree automata is EXPTIME-complete [23]. However, single-type DTDs and the latter automata are incomparable. Indeed, the tree language consisting of the trees $\{a(bc), a(cb)\}$ is not definable by a top-down deterministic tree automaton, while it is by the DTD consisting of the rules $a \to bc + cb$, $b \to \varepsilon$, $c \to \varepsilon$. Conversely, the tree language $\{a(b(c)b(d))\}$ is not definable by a single-type SDTD, but is definable by a top-down deterministic tree automaton.

Theorem 3. *The intersection problem for SDTDst$((+a), w?, (+a)?, (+a)^*)$ is* EXPTIME*-hard.*

4 Inclusion of Regular Expressions

We start our investigation with the inclusion problem for simple regular expressions. As mentioned before, it is PSPACE-complete for general regular expressions. Some tractable cases have been identified in previous work: *(i)* in [1] it is shown that inclusion for $RE(a?, (+a)^*)$ can be solved in linear time. Whether $p \subseteq p_1 + \cdots + p_k$ for $p, p_1, \ldots, p_k$ from $RE(a?, (+a)^*)$ can be checked in quadratic time; and *(ii)* in [17] it is stated that inclusion for $\mathrm{RE}(a, \Sigma, \Sigma^*)$ is in PTIME.

Some of the fragments we defined look so innocent that one might expect their inclusion problem to be tractable. It therefore came as a surprise to find out that even for $RE(a, a?)$ and $RE(a, a^*)$ it is already CONP-complete. Even worse, for $RE(a, (+a)^*)$ we already obtain the maximum complexity, PSPACE-completeness. The latter should be contrasted with the PTIME inclusion for $\mathrm{RE}(a, \Sigma, \Sigma^*)$ obtained in [17], where disjunctions can only be over the complete alphabet. Our results, together with corresponding upper bounds are summarized in the following theorem. Let $\mathrm{RE}(\mathcal{S} - \{(+a)^*, (+w)^*\})$ denote the fragment of $\mathrm{RE}(\mathcal{S})$ where no factor of the form $(+a)^*$ or $(+w)^*$ is allowed.

Theorem 4. *The inclusion problem*

(a) is CONP*-hard for* $RE(a, a^*)$ *and* $RE(a, a?)$*;*
(b) is PSPACE*-hard for* $RE(a, (+a)^*)$*;*
(c) is in CONP *for* $RE(\mathcal{S} - \{(+a)^*, (+w)^*\})$*; and,*
(d) is in PSPACE *for* $RE(\mathcal{S})$*.*

This result does not leave much room for tractable cases. Of course, inclusion is in PTIME for any class of regular expressions for which expressions can be transformed into DFAs in polynomial time. An easy example of such a class is $\mathrm{RE}(a, a^+)$. The following example has probably more importance in practice. Often, the same symbol occurs only a few times in a regular expression of a DTD. As a matter of fact, if we impose a fixed bound k on the number of such occurences, then the inclusion problem becomes tractable. For every k, let $\mathrm{RE}^{\leq k}$ denote the class of all regular expressions where every symbol can occur at most k times.

Theorem 5. *Inclusion for* $RE^{\leq k}$ *is in* PTIME*.*

It should be noted though that the upper bound for the running time is of the form n^k, therefore k should be very small to be really useful.

In the rest of this section, we prove Theorem 4(a). The proofs of parts (b) and (c) can be found in the full version of this paper [15].

We show that for both cases there is a PTIME reduction from VALIDITY of propositional 3DNF formulas. Let $\Phi = C_1 \vee \cdots \vee C_k$ be a propositional formula in 3DNF using variables $\{x_1, \ldots, x_n\}$. In both cases, we are going to construct regular expressions R_1, R_2 such that $L(R_1) \subseteq L(R_2)$ if and only if Φ is valid, i.e., if Φ is true under all truth assignments. More specifically, we encode truth assignments for Φ by strings. The basic idea is to construct R_1 and R_2 such

that $L(R_1)$ contains all such strings and R_2 captures a string w if and only if it represents an assignment which makes Φ true.

Let $U = \#a\$a\$\cdots\$a\#$ (n occurrences of a), be a regular expression describing exactly one string u. We will construct a regular expression W such that the strings of $L(W)$ can be interpreted as truth assignments. More precisely, for each truth assignment A there is a string $w_A \in L(W)$ and for each string $w \in L(W)$ there is a truth assignment A_w. Then we set $R_1 = U^{k-1}WU^{k-1}$ and $R_2 = N^{k-1}F_1 \cdots F_k N^{k-1}$, where for N and the F_i the following conditions should hold:

(i) $\{\varepsilon, u\} \subseteq L(N)$.
(ii) If N^ℓ captures a string $u^\ell z$ or zu^ℓ, for some ℓ, then $z \in L(\#^*)$.
(iii) If A_w makes C_i true then $w \in L(F_i)$. If $w_A \in L(F_i)$ then A makes C_i true.
(iv) $u \in L(F_i)$.
(v) Each string in $L(F_i)$ begins and ends with # and has no other occurences of #.
(vi) Each string in $L(W)$ starts and ends with a # and has no other occurences of #. Further, $\#\# \notin L(W)$.

Assume the existence of such expressions W, N and F_i. We claim that $L(R_1) \subseteq L(R_2)$ if and only if Φ is valid. To prove this claim let first $L(R_1) \subseteq L(R_2)$. Let A be an arbitrary truth assignment for Φ and let $v = u^{k-1}w_A u^{k-1}$. By assumption, as $v \in L(R_1)$ we get $v \in L(R_2)$. By (ii) and (vi), and the form of R_2, w_A can not be captured by a part of the leading or trailing N^{k-1}. By (v) and (vi), it follows that w_A must be covered by some F_i. Hence, C_i is made true by A.

For the other direction, suppose now that Φ is valid. Take an arbitrary $v = u^{k-1}wu^{k-1} \in L(R_1)$ and let A be the truth assigment A_w corresponding to w. Let C_i be a clause of Φ that becomes true under A_w. Due to (iii), we have that $w \in L(F_i)$. Further, as u is captured by each F_i and by N and as $L(N)$ also contains the empty string $u^{k-1}wu^{k-1} \in L(R_2)$. This completes the proof of the claim.

It remains to construct regular expressions W, N and F_i with the required properties. First of all, for both fragments it is easy to define N. For the first fragment, we can take $N = \#^*a^*\$^*a^*\$^*\cdots\$^*a^*\#^*$ (n occurrences of a), for the second, we take '?' in place of '*'.

The construction of the other expressions differs slightly for both cases. Let the expressions r_0, r_1, r_{01} be defined as follows:

(1) $r_0 = b^*a^*$, $r_1 = aa^*b^*a^*$ and $r_{01} = a^*b^*a^*$, in the case of $RE(a, a^*)$, and as
(2) $r_0 = a?$, $r_1 = aa?$ and $r_{01} = a?a?$ for $RE(a, a?)$.

In both cases, these expressions have the properties (INC1)-(INC4) below.

$$a \in L(r_0) \cap L(r_1) \tag{INC1}$$
$$L(r_{01}) \subseteq L(r_0) \cup L(r_1) \tag{INC2}$$
$$L(r_{01}) - L(r_0) \neq \emptyset \tag{INC3}$$
$$L(r_{01}) - L(r_1) \neq \emptyset \tag{INC4}$$

Now we define $W = \#r_{01}\$\cdots\$r_{01}\#$ (n occurrences of r_{01}). With $w = \#w_1\$\cdots\$w_n\# \in L(W)$ we associate a truth assignment A_w as follows: $A_w(x_j) :=$ true, if $w_j \in L(r_1)$; and false otherwise.

Let $z_0 \in L(r_{01}) - L(r_1)$ and $z_1 \in L(r_{01}) - L(r_0)$. They exist by (INC3) and (INC4). For a truth assignment A, let w_A be the string $\#y_1\$\cdots\$y_n\#$, where $y_j = z_1$ if $A(x_j) =$ true and $y_j = z_0$, otherwise.

For each i, we set $F_i = \#e_1\$\cdots\$e_n\#$, where for each $j = 1, \ldots, n$, $e_j = r_0$, $e_j = r_1$ or $e_j = r_{01}$ if x_j occurs negated, unnegated or not at all in C_i. It is easy to show that condition (iii) holds. That the other conditions hold is obvious from the construction.

Remark 2. The CONP lower bounds even hold, if in RE(a, a^*), subexpressions of the form aa^* are forbidden. Indeed, with $r_1 = ac^*a^*b^*a^*$ in case (1), the reduction still holds. Further, in the case of RE$(a, a?)$ two letters suffice: take b and bb in place of \$ and #, respectively.

5 Equivalence of Regular Expressions

In the present section, we merely initiate the research on the equivalence of simple regular expressions. Of course, upper bounds for inclusion induce upper bounds for equivalence, but testing equivalence can be simpler. We show that the problem is in PTIME for $RE(a, a?)$ and $RE(a, a^*)$ by showing that such expressions are equivalent if and only if they have the same normal form (defined below). We conjecture that equivalence remains tractable for larger fragments or even the full fragment of simple regular expressions.

Theorem 6. *The equivalence problem is in* PTIME *for* $RE(a, a?)$*, and* $RE(a, a^*)$*.*

Proof (Sketch). We start by introducing a normal form for RE$(a, a?, a^*)$ expressions. It is often useful to cumulate successive factors of a regular expression that have the same base symbol (or disjunction of base symbols). For such a sequence we only need to know the minimal and maximal number of occurences of the basic expression. By $e[i, j]$ we denote a sequence with at least i and at most j occurences of e. Here, $j = *$ stands for an unlimited number of repetitions. To this end, we write $e[1, 1]$ for e, $e[0, 1]$ for $e?$, and $e[0, *]$ for e^*. By combining successive factors $e[i_1, j_1]$ and $e[i_2, j_2]$ into $e[i_1 + i_2, j_2 + j_2]$ whenever possible, we arrive at the *sequence normal form* of a simple regular expression. E.g., the sequence normal form of $aa?aa?b^*bb?b^*$ is $a[2, 4]b[1, *]$.

For the sake of the proof, this normal form is not sufficient, as e.g., the regular expressions $r_1(a, b) = a[i, *]b[0, *]a[0, *]b[1, *]a[l, *]$ and $r_2(a, b) = a[i, *]b[1, *]a[0, *]b[0, *]a[l, *]$ are equivalent but have different normal forms. Whenever an expression of the form $r_1(a, b)$ occurs it can be replaced by $r_2(a, b)$. The *strong sequence normal form* of an expression r is the expression r' obtained by applying this rule as often as possible. It is easy to see, that r' is unique. For both fragments, it can be shown that two expressions are equivalent if and only if they have the same strong sequence normal form. □

6 Intersection of Regular Expressions

For arbitrary regular expressions, the intersection problem is PSPACE-complete. We show that the problem is NP-hard for the same two innocent fragments $\mathrm{RE}(a, a^*)$ and $\mathrm{RE}(a, a?)$ already studied in Section 4. By $\mathrm{RE}(\mathcal{S} - \{(+w)^*\})$, we denote the fragment of $\mathrm{RE}(\mathcal{S})$ where no factor can be of the form $(+w)^*$; so, factors of the form $(+a)^*$ are allowed. For the latter fragment, we obtain a matching NP-upper bound. The complexity of the full fragment $\mathrm{RE}(\mathcal{S})$ remains open. Our results are summarized in the following theorem:

Theorem 7. *The intersection problem is (a)* NP*-hard for* $RE(a, a^*)$ *and* $RE(a, a?)$*; and (b) in* NP *for* $RE(\mathcal{S} - (+w)^*)$.

Note that Theorem 7(a) makes the difference between the inclusion and intersection problem apparent. Indeed, inclusion of $\mathrm{RE}(\mathcal{S} - \{(+w)^*\})$ is PSPACE-complete, while it is NP-complete for intersection. The latter, however, does not imply that inclusion is always harder than intersection. Indeed, we obtained that for any fixed k inclusion of $\mathrm{RE}^{\leq k}$ is in PTIME. The next theorem shows that for $k = 3$, intersection is PSPACE-hard.

Content models of DTDs are restricted to one-unambiguous regular expressions. For a concrete definition, we refer to [4]. Although inclusion for such expressions is in PTIME, we obtain that the intersection problem is as hard as for arbitrary regular expressions.

Theorem 8. *The intersection problem is* PSPACE*-hard for (a) one-unambiguous regular expressions; and for (b)* $RE^{\leq 3}$.

A tractable fragment is the following:

Theorem 9. *The intersection problem is in* PTIME *for* $RE(a, a^+)$.

7 Conclusion

We revisited the inclusion, equivalence and intersection problem for regular expressions that are common in practical DTDs. Moreover, we showed that for all these problems, the complexities carry over to the corresponding problems for DTDs. For inclusion and equivalence, the complexity bounds for regular expressions also carry over to single-type SDTDs. We left the following complexities open: *(i)* equivalence for $\mathrm{RE}(\mathcal{S})$ or any fragment extending $\mathrm{RE}(a, a^*)$ or $\mathrm{RE}(a, a?)$; and, *(ii)* intersection for $\mathrm{RE}(\mathcal{S})$.

References

1. P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *Proc. of CAV 1998*, pages 305–318, 1998.
2. G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: A practical study. To be presented at WebDB 2004.

3. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.
4. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.
5. A. Brüggemann-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages*, 2(1):81–106, 2000.
6. D. Calvanese, De G. Giacomo, M. Lenzerini, and M. Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.
7. B. Choi. What are real DTDs like? In *WebDB 2002*, pages 43–48, 2002.
8. World Wide Web Consortium. Extensible Markup Language (XML). http://www.w3.org/XML.
9. World Wide Web Consortium. XML Schema. http://www.w3.org/XML/Schema.
10. L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion*. Springer, 2002.
11. H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Transactions on Internet Technology (TOIT)*, 3(2):117–148, 2003.
12. Harry B. Hunt III, Daniel J. Rosenkrantz, and Thomas G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *Journal of Computer and System Sciences*, 12(2):222–268, 1976.
13. D. Kozen. Lower bounds for natural proof systems. In *Proc. FOCS 1977*, pages 254–266. IEEE, 1977.
14. W. Martens and F. Neven. Typechecking top-down uniform unranked tree transducers. In *Proc. ICDT 2003*, pages 64–78, 2003.
15. W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for simple regular expressions: Full version. `http://alpha.luc.ac.be/lucp1436/pubs.html`.
16. M. Marx. XPath with conditional axis relations. In *Proc. EDBT 2004*, pages 477–494, 2004.
17. T. Milo and D. Suciu. Index structures for path expressions. In *Proc. ICDT 1999*, pages 277–295. 1999.
18. T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66(1):66–97, 2003.
19. M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. In *Extreme Markup Languages*, Montreal, Canada, 2001.
20. F. Neven. Automata, logic, and XML. In *Proc. CSL 2002*, pages 2–26. 2002.
21. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. PODS 2000*, pages 35–46. ACM Press, 2000.
22. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.
23. H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60, 1994.
24. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proc. STOC 1973*, pages 1–9, 1973.
25. E. van der Vlist. *Relax NG*. O'Reilly, 2003.
26. V. Vianu. A web odyssey: From Codd to XML. In *Proc. PODS 2001*, pages 1–15, 2001.