Simple Schemas for Unordered XML
Iovka Boneva, Radu Ciucanu, Slawomir Staworko

To cite this version:
Iovka Boneva, Radu Ciucanu, Slawomir Staworko. Simple Schemas for Unordered XML. 16th International Workshop on the Web and Databases (WebDB), Jun 2013, New York, United States. <hal-00824459>

HAL Id: hal-00824459
https://hal.inria.fr/hal-00824459
Submitted on 7 May 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Simple Schemas for Unordered XML

Iovka Boneva  
University of Lille & INRIA, France  
iovka.boneva@inria.fr

Radu Ciucanu  
University of Lille & INRIA, France  
radu.ciucanu@inria.fr

Slawek Staworko  
University of Lille & INRIA, France  
slawomir.staworko@inria.fr

ABSTRACT

We consider unordered XML, where the relative order among siblings is ignored, and propose two simple yet practical schema formalisms: disjunctive multiplicity schemas (DMS), and its restriction, disjunction-free multiplicity schemas (MS). We investigate their computational properties and characterize the complexity of the following static analysis problems: schema satisfiability, membership of a tree to the language of a schema, schema containment, twig query satisfiability, implication, and containment in the presence of schema. Our research indicates that the proposed formalisms retain much of the expressiveness of DTDs without an increase in computational complexity.

1. INTRODUCTION

When XML is used for document-centric applications, the relative order among the elements is typically important e.g., the relative order of paragraphs and chapters in a book. On the other hand, in case of data-centric XML applications, the order among the elements may be unimportant [1]. In this paper we focus on the latter use case. As an example, take a trivialized fragment of an XML document containing the DBLP repository in Figure 1. While the order of the elements title, author, and year may differ from one publication to another, it has no impact on the semantics of the data stored in this semi-structured database.

A schema for XML is a description of the type of admissible documents, typically defining for every node its content model i.e., the children nodes it must, may or cannot contain. For instance, in the DBLP example, we shall require every article to have exactly one title, one year, and one or more author’s. A book may additionally contain one publisher and may also have one or more editor’s instead of author’s. A schema has numerous important uses. For instance, it allows to validate a document against a schema and identify potential errors. A schema also serves as a reference for any user who does not know yet the structure of the XML document and attempts to query or modify its contents.

The Document Type Definition (DTD), the most widespread XML schema formalism for (ordered) XML [6, 13], is essentially a set of rules associating with each label a regular expression that defines the admissible sequences of children. The DTDs are best fitted towards ordered content because they use regular expressions, a formalism that defines sequences of labels. However, when unordered content model needs to be defined, there is a tendency to use over-permissive regular expressions. For instance, the DTD below corresponds to the one used in practice for the DBLP repository:

```
<xml>
</xml>
```

This DTD allows an article to contain any number of title, year, and author elements. A book may also have any number of title, year, author, editor, and publisher elements. These regular expressions are clearly over-permissive because they allow documents that do not follow the intuitive guidelines set out earlier e.g., a document containing an article with two title’s and no author should not be admissible.

While it is possible to capture unordered content models with regular expressions, a simple pumping argument shows that their size may need to be exponential in the number of possible labels of the children. In case of the DBLP repository, this number reaches values up to 12, which basically precludes any practical use of such regular expressions. This suggests that over-permissive regular expressions may be employed for the reasons of conciseness and readability.

The use of over-permissive regular expressions, apart from allowing documents that do not follow the guidelines, has other negative consequences e.g., in static analysis tasks that involve the schema. Take for example the following two twig queries [2, 23]:

```
/dblp/book[author = “C. Papadimitriou”]
/dblp/book[author = “C. Papadimitriou”][title]
```

The first query selects the elements labeled book, children of dblp and having an author containing the text “C. Papadimitriou.” The second query additionally requires that book has a title. Naturally, these two queries should be...
Table 1: Summary of complexity results.

<table>
<thead>
<tr>
<th>Problem of interest</th>
<th>DTD</th>
<th>DMS</th>
<th>disjunction-free DTD</th>
<th>MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema satisfiability</td>
<td>PTIME [9, 21]</td>
<td>PTIME (Prop. 4.3)</td>
<td>PTIME [9, 21]</td>
<td>PTIME (Prop. 4.3)</td>
</tr>
<tr>
<td>Membership</td>
<td>PTIME [9, 21]</td>
<td>PTIME (Prop. 4.3)</td>
<td>PTIME [9, 21]</td>
<td>PTIME (Prop. 4.3)</td>
</tr>
<tr>
<td>Schema containment</td>
<td>PSPACE-c/PTIME [9, 21]</td>
<td>PTIME (Th. 4.2)</td>
<td>coNP-h/PTIME [9, 16]</td>
<td>PTIME (Th. 4.2)</td>
</tr>
<tr>
<td>Query satisfiability†</td>
<td>NP-c [4]</td>
<td>NP-c (Prop. 4.4)</td>
<td>PTIME (Th. 4.7)</td>
<td>PTIME (Th. 4.7)</td>
</tr>
<tr>
<td>Query implication†</td>
<td>EXPTIME-c [19]</td>
<td>EXPTIME-c (Prop. 4.4)</td>
<td>PTIME (Cor. 4.9)</td>
<td>PTIME (Th. 4.7)</td>
</tr>
<tr>
<td>Query containment†</td>
<td>EXPTIME-c [19]</td>
<td>EXPTIME-c (Prop. 4.4)</td>
<td>coNP-c (Cor. 4.9)</td>
<td>coNP-c (Th. 4.8)</td>
</tr>
</tbody>
</table>

† when non-deterministic regular expressions are used. † for twig queries.

We study the complexity of several basic decision problems: schema satisfiability, membership of a tree to the language of a schema, containment of two schemas, twig query satisfiability, implication, and containment in the presence of schema. Table 1 contains the summary of complexity results compared with general DTDs and disjunction-free DTDs. The lower bounds for the decision problems for DMS and MS are generally obtained with easy adaptations of their counterparts for general DTDs and disjunction-free DTDs. To obtain upper bounds we develop several new tools. Dependency graphs for MS and a generalized definition of an embedding of a query help us to reason about query satisfiability, query implication, and query containment in the presence of MS.

An alternative characterization of DMS with characterizing triples of sets is used to reduce the containment of DMS to the containment of the sets of their triples, which can be tested in PTIME. We add that our constructions and results for MS extend easily to disjunction-free DTDs and allow to solve the problems of query implication and query containment, which, to the best of our knowledge, have not been previously studied for disjunction-free DTDs.

Because of space restriction, the proofs of all claims are omitted; they can be found in the appendix of the full version of the paper available at http://arxiv.org/abs/1303.4277.

Related work. Languages of unordered trees can be expressed by logic formalisms or by tree automata. Boneva et al. [7, 8] make a survey on such formalisms and compare their expressiveness. The fundamental difference resides in the kind of constraints that can be expressed for the allowed collections of children for some node. We mention here only formalisms introduced in the context of XML. Presburger automata [22], sheaves automata [11], and the TQL logic [10] allow to express Presburger constraints on the numbers of occurrences of the different symbols among the children of some node. This is also equivalent to considering DTDs under commutative closure [3], similarly to [3]. The consequence of the high expressive power is that the membership problem is NP-complete for an unbounded alphabet [15]. Therefore, these formalisms were not extensively used in practice. Suitable restrictions on Presburger automata and on the TQL logic allow to obtain the same expressiveness as the MSO logic on unordered trees [7, 8]. DMS are strictly less expressive than these MSO-equivalent languages. Static analysis problems involving twig queries were not studied for these languages. Additionally, we believe that DMS are more appropriate to be used as schema languages, as they were designed as such, in particular regarding the more user-friendly DTD-like syntax. As mentioned earlier, unordered content model can also be defined by DTDs defining commutatively-closed sets of ordered trees. An (ordered) tree matches such a DTD if...
2. PRELIMINARIES

Throughout this paper we assume an alphabet $\Sigma$ which is a finite set of symbols.

**Trees.** We model XML documents with unordered labeled trees. Formally, a tree $t$ is a tuple $(N_t, \text{root}_t, \text{lab}_t, \text{child}_t)$, where $N_t$ is a finite set of nodes, $\text{root}_t \in N_t$ is a distinguished root node, $\text{lab}_t : N_t \rightarrow \Sigma$ is a labeling function, and $\text{child}_t \subseteq N_t \times N_t$ is the parent-child relation. We assume that the relation $\text{child}_t$ is acyclic and require every non-root node to have exactly one predecessor in this relation. By Tree we denote the set of all finite trees.

**Queries.** We work with the class of twig queries, which are essentially unordered trees whose nodes may be additionally labeled with a distinguished wildcard symbol $\bullet \not\in \Sigma$ and that use two types of edges, child (//) and descendant (\///), corresponding to the standard XPath axes. Note that the semantics of the edge is that of a proper descendant (and not that of descendant-or-self). Formally, a twig query $q$ is a tuple $(N_q, \text{root}_q, \text{lab}_q, \text{child}_q, \text{desc}_q)$, where $N_q$ is a finite set of nodes, $\text{root}_q \in N_q$ is the root node, $\text{lab}_q : N_q \rightarrow \Sigma \cup \{\bullet\}$ is a labeling function, $\text{child}_q \subseteq N_q \times N_q$ is a set of child edges, and $\text{desc}_q \subseteq N_q \times N_q$ is a set of descendant edges. We assume that $\text{child}_q \cap \text{desc}_q = \emptyset$ and that the relation $\text{child}_q \cup \text{desc}_q$ is acyclic and we require every non-root node to have exactly one predecessor in this relation. By Twig we denote the set of all twig queries. Twig queries are often presented using the abbreviated XPath syntax [23] e.g., the query $q_0$ in Figure 2(b) can be written as $r/\text{a}[b]/\text{a}$.

**Embeddings.** We define the semantics of twig queries using the notion of embedding which is essentially a mapping of nodes of a query to the nodes of a tree that respects the semantics of the edges of the query. Formally, for a query $q \in \text{Twig}$ and a tree $t \in \text{Tree}$, an embedding $\lambda : N_q \rightarrow N_t$ such that:

1. $\lambda(\text{root}_q) = \text{root}_t$.
2. For every $(n, n') \in \text{child}_q$, $(\lambda(n), \lambda(n')) \in \text{child}_t$.
3. For every $(n, n') \in \text{desc}_q$, $(\lambda(n), \lambda(n')) \in (\text{child}_t)^+$ (the transitive closure of $\text{child}_t$).
4. For every $n \in N_q$, lab$_q(n) = \bullet$ or lab$_q(n) = \text{lab}_t(\lambda(n))$.

If there exists an embedding from $q$ to $t$ we say that $t satisfies$ q and we write $t \mid q$. By $L(q)$ we denote the set of all the trees satisfying $q$. Note that we do not require the embedding to be injective i.e., two nodes of the query may be mapped to the same node of the tree. Figure 3 presents all embeddings of the query $q_0$ in the tree $t_0$ from Figure 2.

![Figure 2: A tree and a twig query.](image)

![Figure 3: Embeddings of $q_0$ in $t_0$.](image)

3. MULTIPLICITY SCHEMAS

A multiplicity is an element from the set $\{\ast, +, ?, 0, 1\}$. We define the function $[\ast]$ mapping multiplicities to sets of natural numbers. More precisely:

$[\ast] = \{i \in \mathbb{N} \mid i \geq 0\}$, $[+] = \{i \in \mathbb{N} \mid i \geq 1\}$, $[?] = \{0, 1\}$, $[1] = \{1\}$, $[\emptyset] = \{0\}$.

Given a symbol $a \in \Sigma$ and a multiplicity $M$, the language of $a^M$, denoted $L(a^M)$, is $\{a^i \mid i \in [M]\}$. For example, $L(a^1) = \{a, aa, \ldots\}$, $L(\emptyset) = \{\varepsilon\}$, and $L(1) = \{\varepsilon, c\}$. 

Unordered words. An unordered word is essentially a multiset of symbols i.e., a function $w : \Sigma \rightarrow \mathbb{N}_0$ mapping symbols from the alphabet to natural numbers, and we call the number $w(a)$ the number of occurrences of the symbol $a$ in $w$. We also write $a \in w$ as a shorthand for $w(a) \neq 0$. An empty word $\varepsilon$ is an unordered word that has 0 occurrences of every symbol i.e., $\varepsilon(a) = 0$ for every $a \in \Sigma$. We often use a simple representation of unordered words, writing each symbol in the alphabet the number of times it occurs in the unordered word. For example, when the alphabet is $\Sigma = \{a, b, c\}$, $w_0 = aacc$ stands for the function $w_0(a) = 3$, $w_0(b) = 0$, and $w_0(c) = 2$.

The (unordered) concatenation of two unordered words $w_1$ and $w_2$ is defined as the multiset union $w_1 \cup w_2$ i.e., the function defined as $(w_1 \cup w_2)(a) = w_1(a) + w_2(a)$ for all $a \in \Sigma$. For instance, $aaacc \cup abc = aaaaabcccc$. Note that $\varepsilon$ is the identity element of the unordered concatenation $\varepsilon \uplus w = w \uplus \varepsilon = w$ for all unordered word $w$. Also, given an unordered word $w$, by $w^1$ we denote the concatenation $w \ldots w \ldots w$ (1 times).

A language is a set of unordered words. The unordered concatenation of two languages $L_1$ and $L_2$ is a language $L_1 \uplus L_2 = \{w_1 \cup w_2 \mid w_1 \in L_1, w_2 \in L_2\}$. For instance, if $L_1 = \{a, aac\}$ and $L_2 = \{ac, b, \varepsilon\}$, then $L_1 \uplus L_2 = \{a, b, aac, aabc, aacc\}$.
A disjunctive multiplicity expression E is:
\[ E := D_1^{M_1} \| \ldots \| D_n^{M_n}, \]
where for all \(1 \leq i \leq n\), \(M_i\) is a multiplicity and each \(D_i\) is:
\[ D_i := a_1^{M_{i1}} \| \ldots \| a_n^{M_{in}}, \]
where for all \(1 \leq j \leq k\), \(M_j\) is a multiplicity and \(a_j \in \Sigma\). Moreover, we require that every symbol \(a \in \Sigma\) is present at most once in a disjunctive multiplicity expression. For instance, \((a \mid b) \| (c \mid d)\) is a disjunctive multiplicity expression, but \((a \mid b) \| (a \mid d)\) is not because \(a\) appears twice. A disjunction-free multiplicity expression is an expression which uses no disjunction symbol “\(|\)”, i.e., an expression of the form \(a_1^{M_1} \| \ldots \| a_n^{M_n}\), where for all \(1 \leq i \leq k\), the \(a_i\)’s are pairwise distinct symbols in the alphabet and the \(M_i\)’s are multiplicities.

The language of a disjunctive multiplicity expression is:
\[ L(E) = \{a^{b_1}c_1d_1 \mid i,j,k \in \mathbb{N}, i \geq 1, j + k = 1, \ell \leq 1\}. \]

Next, we formally define the proposed schema formalisms.

**Definition 3.1** A disjunctive multiplicity schema (DMS) is a tuple \(S = (\text{root}_S, R_S)\), where \(\text{root}_S \in \Sigma\) is a designated root label and \(R_S\) maps symbols in \(\Sigma\) to disjunctive multiplicity expressions. By DMS we denote the set of all disjunctive multiplicity schemas. A disjunction-free multiplicity schema (MS) \(S = (\text{root}_S, R_S)\) is a restriction of the DMS, where \(R_S\) maps symbols in \(\Sigma\) to disjunction-free multiplicity expressions. By MS we denote the set of all disjunction-free multiplicity schemas.

To define satisfiability of a DMS (or MS) \(S\) by a tree \(t\) we first define the unordered word \(ch^n_t(a)\) of children of a node \(a \in N_t\), of \(t\) i.e., \(ch^n_t(a) = \{m \in N_t \mid (n,m) \in \text{child}_t\} \land (\text{lab}_t(m) = a\}\)

Now, a tree \(t\) satisfies \(S\), in symbols \(t \models S\), if \(\text{lab}_t(\text{root}_t) = \text{root}_S\) and for any node \(n \in N_t\), \(ch^n_t(a) \subseteq R_S(\text{lab}_t(n))\). By \(L(S) \subseteq \text{Tree}\) we denote the set of all the trees satisfying \(S\).

In the sequel, we represent a schema \(S = (\text{root}_S, R_S)\) as a set of rules of the form \(a \rightarrow R_S(a)\), for any \(a \in \Sigma\). If \(L(R_S(a)) = \epsilon\), then we write \(a \rightarrow \epsilon\) or we simply omit writing such a rule.

**Example 3.2** We present schemas \(S_1, S_2, S_3, S_4\) illustrating the formalisms defined above. The tree \(t_0\) to from Figure 2(a) satisfies only \(S_1\) and \(S_3\).

### 4. Static Analysis

We first define the problems of interest and we formally state the corresponding decision problems parameterized by the class of schema and, when appropriate, by a class of queries.

- **Schema satisfiability** – checking if there exists a tree satisfying the given schema:

\[ \text{SAT}_S = \{S \in \mathcal{S} \mid \exists t \in \text{Tree}. \ t \models S\}. \]

- **Membership** – checking if the given tree satisfies the given schema:

\[ \text{MEMB}_S = \{(S, t) \in \mathcal{S} \times \text{Tree} \mid t \models S\}. \]

- **Schema containment** – checking if every tree satisfying one given schema satisfies another given schema:

\[ \text{CNT}_S = \{(S_1, S_2) \in \mathcal{S} \times \mathcal{S} \mid L(S_1) \subseteq L(S_2)\}. \]

- **Query satisfiability by schema** – checking if there exists a tree that satisfies the given schema and the given query:

\[ \text{SAT}_{S,Q} = \{(S, q) \in \mathcal{S} \times \mathcal{Q} \mid \exists t \in L(S). \ t \models q\}. \]

- **Query implication by schema** – checking if every tree satisfying the given schema also satisfies the given query:

\[ \text{IMPL}_{S,Q} = \{(p, q, S) \in \mathcal{Q} \times \mathcal{Q} \times \mathcal{S} \mid \forall t \in L(S). \ t \models p \Rightarrow t \models q\}. \]

We next study these decision problems for DMS and MS.

### 4.1 Disjunctive multiplicity schema

In this section we present the complexity results for DMS. We first introduce an alternative representation of schemas used to establish the complexity of schema containment. Recall that \(a \in w\) means that \(w(a) \neq 0\). Given a disjunctive multiplicity expression \(E\), we define the (characterizing) triple \((C_E, N_E, P_E)\) of \(E\) consisting of the following sets:

- **The conflicting pairs of siblings** \(C_E\) consisting of pairs of symbols in \(\Sigma\) such that \(E\) defines no word using both symbols simultaneously:

\[ C_E = \{(a_1, a_2) \in \Sigma \times \Sigma \mid \not\exists w \in L(E). \ a_1 \in w \land a_2 \in w\}. \]

- **The extended cardinality map** \(N_E\) captures for each symbol in the alphabet the possible numbers of its occurrences in the unordered words defined by \(E\):

\[ N_E = \{(a, w(a)) \in \Sigma \times \mathbb{N} \mid w \in L(E)\}. \]

- **The sets of required symbols** \(P_E\) which captures symbols that must be present in every word; essentially, a set of symbols \(X\) belongs to \(P_E\) if every word defined by \(E\) contains at least one element from \(X\):

\[ P_E = \{X \subseteq \Sigma \mid \forall w \in L(E). \ \exists a \in X. \ a \in w\}. \]

As an example we take \(E_0 = a^* \| (b \mid c) \| d^2\). Because \(P_E\) is closed under supersets, we list only its minimal elements:

\[ C_{E_0} = \{(b, c), (c, b)\}, \quad P_{E_0} = \{\{a\}, \{b, c\}, \ldots\}, \quad N_{E_0} = \{(0, 0), (b, 1), (c, 0), (c, 1), (d, 0), (d, 1), (c, 1), (a, 1), (a, 2), \ldots\}. \]
The triples allow us to capture the containment of disjunctive multiplicity expressions:

**Lemma 4.1** For all disjunctive multiplicity expressions $E_1$ and $E_2$, $I(E_2) \subseteq I(E_1)$ iff $C_{E_1} \subseteq C_{E_2}$, $N_{E_1} \subseteq N_{E_2}$, and $P_{E_1} \subseteq P_{E_2}$.

The above lemma also shows that two equivalent DMS yield the same triples and hence the triple $(C_E, N_E, P_E)$ can be viewed as the normal form of a given expression $E$. We point out that $N_E$ may be infinite but it can be represented in a compact manner using multiplicities: for all letter $a$, the set $\{x \in \mathbb{N} \mid (a, x) \in N_E\}$ is representable by a multiplicity. Also, $P_E$ may be exponential in the size of $\Sigma$ but it can be represented with its $\Sigma$-minimal elements. Note that $C_E$ has a number of elements quadratic in the number of letters, but allows for a representation linear in the size of $\Sigma$. These compact representations allow to easily test inclusion. Using Lemma 4.1, this allows us to establish one of our main results:

**Theorem 4.2** CNT$_{DMS}$ is in PTIME.

Next we present complexity results for satisfiability and membership. A streaming algorithm processes an XML document in a single pass, and it is earliest since it outputs its result at the earliest point. For a tree $t$, $\text{height}(t)$ is the height of $t$ defined in the usual way. We employ the standard RAM model and assume that subsequent natural numbers are used as labels in $\Sigma$.

**Proposition 4.3** Checking satisfiability of a DMS $S$ can be done in time $O(|\Sigma|^2)$. There exists an earliest streaming algorithm that checks membership of a tree $t$ in a DMS $S$ in time $O(|\Sigma| \times |t| + |\Sigma|^2)$ and using space $O(\text{height}(t) \times |\Sigma| + |\Sigma|^2)$.

For satisfiability, the algorithm performs a preprocessing in time $O(|\Sigma|^2)$, and then performs a simple process based on dynamic programming to check that $S$ is satisfiable. For checking membership of a tree $t$ to the language of $S$, the input tree $t$ is given in XML format. The algorithm works for any arbitrary ordering of sibling nodes. In a preprocessing stage, the algorithm constructs (compact representations of) the triples of expressions used by the schema $S$ and requires $O(|\Sigma|^2)$ space. During the execution the algorithm maintains a stack whose height is the depth of the currently visited node. The algorithm stores on the stack a vector mapping $a \in \Sigma$ to a value among 0, 1, or $2^+$ indicating that the current node has none, one or more occurrences of the symbol $a$ among its children. Naturally, the bound on space required for stack operations is $O(\text{height}(t) \times |\Sigma|)$. The algorithm is earliest and rejects a tree as soon as the opening tag is read for nodes that violate either some conflicting pair or the allowed cardinality.

We end the section with three complexity results that follow from known facts.

**Proposition 4.4** SAT$_{DMS,Twig}$ is NP-complete. IMPL$_{DMS,Twig}$ and CNT$_{DMS,Twig}$ are EXPTIME-complete.

**Proof sketch.** Query satisfiability for DTDs is NP-complete [4] and can be adapted to DMS. Complexity of query implication and query containment follow from the EXPTIME-completeness of twig query containment in presence of DTDs [19]. □

---

### 4.2 Disjunction-free multiplicity schema

In this section we present the complexity results for MS. Although query satisfiability and query implication are tractable for DMS, these problems become tractable for MS because they can be reduced to testing embedding of queries in some dependency graphs that we define in the sequel. Recall that MS use expressions of the form $a^1_1 b^1_1 \ldots a^n_1 n^1$.

**Definition 4.5** Given an MS $S = (root_S, R_S)$, the dependency graph of $S$ is a directed rooted graph $G_S = (\Sigma, root_S, E_S)$ with the node set $\Sigma$, where $root_S$ is the distinguished root node and $(a, b) \in E_S$ if $R_S(a) = \ldots b M \ldots$ and $M \in \{\ast, +, ?, !\}$. Furthermore, the edge $(a, b)$ is called nullable if $0 \in [M]$ (i.e., $M$ is $\ast$ or $?$), otherwise $(a, b)$ is called non-nullable (i.e., $M$ is $+$ or $1$). The universal dependency graph of an MS $S$ is the subgraph $G^U_S$ containing only the non-nullable edges.

In Figure 4 we present the dependency graphs for the schema $S_5$, containing the rules $r \rightarrow a^1 \parallel b^1$, $a \rightarrow b^1$, $b \rightarrow e$.

![Dependency graph G_{S5} and universal dependency graph G^U_{S5} for schema S5.](image)

An MS $S$ is pruned if $G^U_S$ is acyclic. We observe that any MS has an equivalent pruned version which can be constructed in PTIME by removing the rules for the labels from which a cycle can be reached in the universal dependency graph. Note that a schema is satisfiable iff no cycle can be reached from its root in the universal dependency graph. From now on, we assume w.l.o.g. that all the MS that we manipulate are pruned.

We generalize the notion of embedding as a mapping of the nodes of a query $q$ to the nodes of a rooted graph $G = (\Sigma, root, E)$, which can be either a dependency graph or a universal dependency graph. Formally, an **embedding** of $q$ in $G$ is a function $\lambda : N_q \rightarrow \Sigma$ such that:

1. $\lambda(root_q) = root$,
2. for every $(n, n') \in \text{child}_q (\lambda(n), \lambda(n')) \in E$,
3. for every $(n, n') \in \text{desc}_q (\lambda(n), \lambda(n')) \in E^+$ (the transitive closure of $E$),
4. for every $n \in N_q, \text{lab}_q(n) = \ast$ or $\text{lab}_q(n) = \lambda(n)$.

If there exists an embedding from $q$ to $G$, we write $G \leq q$. The dependency graphs and embeddings capture satisfiability and implication of queries by MS.

**Lemma 4.6** For a twig query $q$ and an MS $S$ we have: 1) $q$ is satisfiable by $S$ iff $G_S \leq q$, 2) $q$ is implied by $S$ iff $G^U_S \leq q$.

Furthermore, testing the embedding of a query in a graph can be done in polynomial time with a simple bottom-up algorithm. Hence,

**Theorem 4.7** SAT$_{MS,Twig}$ and IMPL$_{MS,Twig}$ are in PTIME.

The intractability of the containment of twig queries [17] implies the coNP-hardness of the containment of twig queries in the presence of MS. Proving the membership of the problem to coNP is, however, not trivial. Given an instance $(p, q, S)$, the set of all the trees satisfying $p$ and $S$ can be
characterized with a set \( G(p, S) \) containing an exponential number of polynomially-sized graphs and \( p \) is contained in \( q \) in the presence of \( S \) iff the query \( q \) can be embedded into all the graphs in \( G(p, S) \). This condition is easily checked by a non-deterministic Turing machine.

**Theorem 4.8** \( \text{CNT}_{MS, Twig} \) is coNP-complete.

We also point out that the results are easily adapted to disjunction-free DTDs, which allows us to state results which, to the best of our knowledge, are novel.

**Corollary 4.9** \( \text{IMPL}_{dix-free-DTD, Twig} \) is in \( \text{PTIME} \) and \( \text{CNT}_{dix-free-DTD, Twig} \) is coNP-complete.

5. **EXPRESSIVENESS OF DMS**

We compare the expressive power of DMS and DTDs with focus on schemas used in real-life applications. First, we introduce a simple tool for comparing regular expressions with disjunctive multiplicity expressions, and by extension, DTDs with DMS. For a regular expression \( R \), the language \( L(R) \) of unordered words is obtained by removing the relative order of symbols from every ordered word defined by \( R \). A disjunctive multiplicity expression \( E \) captures \( R \) if \( L(E) = L(R) \). A DMS \( S \) captures a DTD \( D \) if for every symbol the disjunctive multiplicity expression on the rhs of a rule in \( S \) captures the regular expression on the rhs of the corresponding rule in \( D \). We believe that this simple comparison is adequate because if a DTD is to be used in a data-centric application, then supposedly the order between siblings is not important. Therefore, a DMS that captures a given DTD defines basically the same type of admissible documents, without imposing any order among siblings. Naturally, if we use the above notion to compare the expressive powers of DTDs and DMS, DTDs are strictly more expressive than DMS.

We use the comparison on the XMark [20] benchmark and the University of Amsterdam XML Web Collection [13]. We find that all 77 regular expressions of the XMark benchmark are captured by DMS rules, and among them 76 by MS rules. As for the DTDs found in the University of Amsterdam XML Web Collection, 84% of regular expressions (with repetitions discarded) are captured by DMS rules and among them 74.6% by MS rules. Moreover, 55.5% of full DTDs in the collection are captured by DMS and among them 45.8% by MS. Note that these figures should be interpreted with caution, as we do not know which of the considered DTDs were indeed intended for data-centric applications. We believe, however, that these numbers give a generally positive answer to the question of how much the expressive power of DTDs the proposed schema formalisms, DMS and MS, retain.

6. **CONCLUSIONS AND FUTURE WORK**

We have studied the computational properties and the expressive power of new schema formalisms, designed for unordered XML: the disjunctive multiplicity schema (DMS) and its restriction, the disjunction-free multiplicity schema (MS). DMS and MS can be seen as DTDs by restricting classes of regular expressions and interpreted under commutative closure to define unordered content models. These restrictions allow on the one hand to maintain a relatively low computational complexity of basic static analysis problems while retaining a significant part of expressive power of DTDs.

An interesting question remains open: are these the most general restrictions that allow to maintain a low complexity profile? We believe that the answer to this question is negative and intend to identify new practical features that could be added to DMS and MS. One such feature are numeric occurrences [14] of the form \( a^{n,m} \) that generalize multiplicities by requiring the presence of at least \( n \) and no more than \( m \) elements \( a \). It would also be interesting to see to what extent our results can be used to propose hybrid schemas that allow to define ordered content for some elements and unordered model for others.

7. REFERENCES


