Theory Comput Syst (2015) 57:337–376
DOI 10.1007/s00224-014-9593-1

Schemas for Unordered XML on a DIME

Iovka Boneva · Radu Ciucanu · Sławek Staworko

Published online: 16 December 2014

Abstract We investigate schema languages for unordered XML having no relative order among siblings. First, we propose *unordered regular expressions* (UREs), essentially regular expressions with *unordered concatenation* instead of standard concatenation, that define languages of unordered words to model the allowed content of a node (i.e., collections of the labels of children). However, unrestricted UREs are computationally too expensive as we show the intractability of two fundamental decision problems for UREs: membership of an unordered word to the language of a URE and containment of two UREs. Consequently, we propose a practical and tractable restriction of UREs, *disjunctive interval multiplicity expressions* (DIMEs). Next, we employ DIMEs to define languages of unordered trees and propose two schema languages: *disjunctive interval multiplicity schema* (DIMS), and its restriction, *disjunction-free interval multiplicity schema* (IMS). We study the complexity of the following static analysis problems: schema satisfiability, membership of a tree to the language of a schema, schema containment, as well as twig query satisfiability, implication, and containment in the presence of schema. Finally, we study the expressive power of the proposed schema languages and compare them

A preliminary version of this article has appeared in the Proceedings of the 16th International Workshop on the Web and Databases (WebDB), 2013 [10].

I. Boneva · R. Ciucanu (✉) · S. Staworko
University of Lille, INRIA, Lille, France
e-mail: radu.ciucanu@inria.fr

I. Boneva
e-mail: iovka.boneva@inria.fr

S. Staworko
e-mail: slawomir.staworko@inria.fr

with yardstick languages of unordered trees (FO, MSO, and Presburger constraints) and DTDs under commutative closure. Our results show that the proposed schema languages are capable of expressing many practical languages of unordered trees and enjoy desirable computational properties.

Keywords Schemas for XML · Unordered XML · Regular expressions · Twig queries · Semi-structured data

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 Fig. 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.

Typically, a *schema* for XML defines for every node its *content model* i.e., the children nodes it must, may, and cannot contain. For instance, in the DBLP example, one would require every article to have exactly one title, one year, and one or more authors. A book may additionally contain one publisher and may also have one or more editors instead of authors. 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 a user who does not know yet the structure of the XML document and attempts to query or modify its content.

The *Document Type Definition* (DTD), the most widespread XML schema formalism for (ordered) XML [8, 23], 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 for 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.

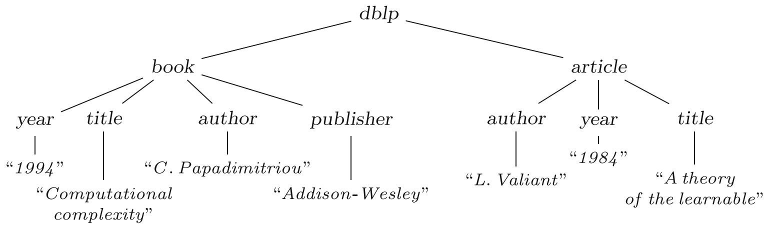


Fig. 1 A trivialized DBLP repository

For instance, the DTD below corresponds to the one used in practice for the DBLP repository:[1]

$$\begin{aligned} \mathit{dblp} &\to (\mathit{article} \mid \mathit{book})^* \\ \mathit{article} &\to (\mathit{title} \mid \mathit{year} \mid \mathit{author})^* \\ \mathit{book} &\to (\mathit{title} \mid \mathit{year} \mid \mathit{author} \mid \mathit{editor} \mid \mathit{publisher})^* \end{aligned}$$

This DTD allows an article to contain any number of titles, years, and authors. A book may also have any number of titles, years, authors, editors, and publishers. These regular expressions are clearly over-permissive because they allow XML documents that do not follow the intuitive guidelines set out earlier e.g., an XML document containing an article with two titles and no should not be valid.

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, a consideration of great practical importance.

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 [3, 47]:

$$/\mathit{dblp}/\mathit{book}[\mathit{author} = \text{``}\mathit{C.\,Papadimitriou}\text{''}]$$

$$/\mathit{dblp}/\mathit{book}[\mathit{author} = \text{``}\mathit{C.\,Papadimitriou}\text{''}][\mathit{title}]$$

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 equivalent because every book should have a title. However, the DTD above does not capture properly this requirement, and consequently the two queries are not equivalent w.r.t. this DTD.

In this paper, we investigate schema languages for unordered XML. First, we study languages of *unordered words*, where an unordered word can be seen as a multiset of symbols. We consider *unordered regular expressions* (UREs), which are essentially regular expressions with *unordered concatenation* "$\|$" instead of standard concatenation. The unordered concatenation can be seen as union of multisets, and consequently, the star "$*$" can be seen as the Kleene closure of unordered languages.

Similarly to a DTD which associates to each label a regular expression to define its (ordered) content model, an unordered schema uses UREs to define for each label

[1] http://dblp.uni-trier.de/xml/dblp.dtd

its unordered content model. For instance, take the following schema (satisfied by the tree in Fig. 1):

$$\begin{aligned} \mathit{dblp} &\to \mathit{article}^* \parallel \mathit{book}^* \\ \mathit{article} &\to \mathit{title} \parallel \mathit{year} \parallel \mathit{author}^+ \\ \mathit{book} &\to \mathit{title} \parallel \mathit{year} \parallel \mathit{publisher}^? \parallel (\mathit{author}^+ \mid \mathit{editor}^+) \end{aligned}$$

The above schema uses UREs and captures the intuitive requirements for the DBLP repository. In particular, an article must have exactly one title, exactly one year, and at least one author. A book may additionally have a publisher and may have one or more editors instead of authors. Note that, unlike the DTD defined earlier, this schema does not allow documents having an article with several titles or without any author.

Using UREs is equivalent to using DTDs with regular expressions interpreted under the *commutative closure* [4, 34]: essentially, a word matches the commutative closure of a regular expression if there exists a permutation of the word that matches the regular expression in the standard way. Deciding this problem is known to be NP-complete [26] for arbitrary regular expressions. We show that the problem of testing the membership of an unordered word to the language of a URE is NP-complete even for a restricted subclass of UREs that allows unordered concatenation and the option operator "?" only. Not surprisingly, testing the containment of two UREs is also intractable. These results are of particular interest because they are novel and do not follow from complexity results for regular expressions, where the order plays typically an essential role [31, 46]. Consequently, we focus on finding restrictions rendering UREs tractable and capable of capturing practical languages in a simple and concise manner.

The first restriction is to disallow repetitions of a symbol in a URE, thus banning expressions of the form $a \parallel a^?$ because the symbol a is used twice. Instead we add general interval multiplicities $a^{[1,2]}$ which offer a way to specify a range of occurrences of a symbol in an unordered word without repeating a symbol in the URE. While the complexity of the membership of an unordered word to the language of a URE with interval multiplicities and without symbol repetitions has recently been shown to be in PTIME [11], testing containment of two such UREs remains intractable. We, therefore, add limitations on the nesting of the disjunction and the unordered concatenation operators and the use of intervals, which yields the proposed class of *disjunctive interval multiplicity expressions* (DIMEs). DIMEs enjoy good computational properties: both the membership and the containment problems become tractable. Also, we believe that despite the imposed restriction DIMEs remain a practical class of UREs. For instance, all UREs used in the schema for the DBLP repository above are DIMEs.

Next, we employ DIMEs to define languages of unordered trees and propose two schema languages: *disjunctive interval multiplicity schema* (DIMS), and its restriction, *disjunction-free interval multiplicity schema* (IMS). Naturally, the above schema for the DBLP repository is a DIMS. 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. We present in Table 1 a summary of the complexity results

Table 1 Summary of complexity results

Problem of interest	*DTD*	*DIMS*	*disj.-free DTD*	*IMS*
Schema satisfiability	PTIME [14, 40]	PTIME (Pr. 1)	PTIME [14, 40]	PTIME (Pr. 1)
Membership	PTIME [14, 40]	PTIME (Pr. 2)	PTIME [14, 40]	PTIME (Pr. 2)
Schema containment	PSPACE-c† [40] PTIME [14]	PTIME (Pr. 1)	coNP-h† [30] PTIME [14]	PTIME (Pr. 1)
Query ‡ satisfiability	NP-c [5]	NP-c (Pr. 3)	PTIME [5]	PTIME (Th. 5)
Query ‡ implication	EXPTIME-c [35]	EXPTIME-c (Pr. 4)	PTIME (Th. 7)	PTIME (Th. 5)
Query ‡ containment	EXPTIME-c [35]	EXPTIME-c (Pr. 4)	coNP-c (Th. 7)	coNP-c (Th. 6)

† When non-deterministic regular expressions are used

‡ For twig queries

and we observe that DIMSs and IMSs enjoy the same computational properties as general DTDs and disjunction-free DTDs, respectively.

The lower bounds for the decision problems for DIMSs and IMSs are generally obtained with easy adaptations of their counterparts for general DTDs and disjunction-free DTDs. To obtain the upper bounds we develop several new tools. We propose to represent DIMEs with *characterizing tuples* that can be efficiently computed and allow deciding in polynomial time the membership of a tree to the language of a DIMS and the containment of two DIMSs. Also, we develop *dependency graphs* for IMSs and a generalized definition of an *embedding* of a query. These two tools help us to reason about query satisfiability, query implication, and query containment in the presence of IMSs. Our constructions and results for IMSs allow also to characterize the complexity of query implication and query containment in the presence of disjunction-free DTDs, which, to the best of our knowledge, have not been previously studied.

Finally, we compare the expressive power of the proposed schema languages with yardstick languages of unordered trees (FO, MSO, and Presburger constraints) and DTDs under commutative closure. We show that the proposed schema languages are capable of expressing many practical languages of unordered trees.

It is important to mention that this paper is a substantially extended version of a preliminary work presented in [10]. More precisely, in this paper we show novel intractability results for some subclasses of unordered regular expressions and we extend the expressibility of the tractable subclasses. While in [10] we have considered only simple multiplicities ($*$, $+$, ?), in this paper we deal with arbitrary interval multiplicities of the form $[n, m]$.

Organization In Section 2 we introduce some preliminary notions. In Section 3 we study the reasons of intractability of unordered regular expressions while in Section 4 we present the tractable subclass of *disjunctive interval multiplicity expressions* (DIMEs). In Section 5 we define two schema languages: the *disjunctive interval multiplicity schemas* (DIMSs) and its restriction, the *disjunction-free interval multiplicity schemas* (IMSs), and the related problems of interest. In Sections 6 and 7 we analyze the complexity of the problems of interest for DIMSs and IMSs, respectively.

In Section 8 we discuss the expressiveness of the proposed formalisms. In Section 9 we present related work. In Section 10 we summarize our results and outline further directions.

2 Preliminaries

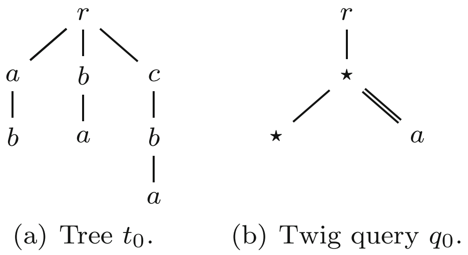
Throughout this paper we assume an alphabet Σ that is a finite set of symbols. We also assume that Σ has a total order $<_\Sigma$ that can be tested in constant time.

Trees We model XML documents with unordered labeled trees. Formally, a *tree* t is a tuple $(N_t, root_t, lab_t, child_t)$, where N_t is a finite set of nodes, $root_t \in N_t$ is a distinguished root node, $lab_t : N_t \to \Sigma$ is a labeling function, and $child_t \subseteq N_t \times N_t$ is the parent-child relation. We assume that the relation $child_t$ is acyclic and require every non-root node to have exactly one predecessor in this relation. By *Tree* we denote the set of all 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 $\star \notin \Sigma$ 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 $(N_q, root_q, lab_q, child_q, desc_q)$, where N_q is a finite set of nodes, $root_q \in N_q$ is the root node, $lab_q : N_q \to \Sigma \cup \{\star\}$ is a labeling function, $child_q \subseteq N_q \times N_q$ is a set of child edges, and $desc_q \subseteq N_q \times N_q$ is a set of descendant edges. We assume that $child_q \cap desc_q = \emptyset$ and that the relation $child_q \cup desc_q$ 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 [47] e.g., the query q_0 in Fig. 2a can be written as $r/\star[\star]//a$.

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 $q \in Twig$ and a tree $t \in Tree$, an *embedding* of q in t is a function $\lambda : N_q \to N_t$ such that:

1. $\lambda(root_q) = root_t$,
2. for every $(n, n') \in child_q$, $(\lambda(n), \lambda(n')) \in child_t$,



(a) Tree t_0. (b) Twig query q_0.

Fig. 2 A tree and a twig query

3. for every $(n, n') \in desc_q$, $(\lambda(n), \lambda(n')) \in (child_t)^+$ (the transitive closure of $child_t$),
4. for every $n \in N_q$, $lab_q(n) = \star$ or $lab_q(n) = lab_t(\lambda(n))$.

We write $t \preccurlyeq q$ if there exists an embedding of q in t. Later on, in Section 7.2 we generalize this definition of embedding as a tool that permits us characterizing the problems of interest.

As already mentioned, we use the notion of embedding to define the semantics of twig queries. In particular, we say that t *satisfies* q if there exists an embedding of q in t and we write $t \models q$. By $L(q)$ we denote the set of all 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 Fig. 2.

Unordered Words An *unordered word* is essentially a multiset of symbols i.e., a function $w : \Sigma \to \mathbb{N}_0$ mapping symbols from the alphabet to natural numbers. We call $w(a)$ 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 ε 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 $\Sigma = \{a, b, c\}$, $w_0 = aaacc$ stands for the function $w_0(a) = 3$, $w_0(b) = 0$, and $w_0(c) = 2$. Additionally, we may write $w_0 = a^3c^2$ instead of $w_0 = aaacc$.

We use unordered words to model collections of children of XML nodes. As it is usually done in the context of XML validation [41, 42], we assume that the XML document is encoded in unary i.e., every node takes the same amount of memory. Thus, we use a unary representation of unordered words, where each occurrence of a symbol occupies the same amount of space. However, we point out that none of the results presented in this paper changes with a binary representation. In particular, the intractability of the membership of an unordered word to the language of a URE (Theorem 1) also holds with a binary representation of unordered words.

Consequently, the *size* of an unordered word w, denoted $|w|$, is the sum of the numbers of occurrences in w of all symbols in the alphabet. For instance, the size of $w_0 = aaacc$ is $|w_0| = 5$.

The (*unordered*) *concatenation* of two unordered words w_1 and w_2 is defined as the multiset union $w_1 \uplus w_2$ i.e., the function defined as $(w_1 \uplus w_2)(a) = w_1(a) + w_2(a)$ for every $a \in \Sigma$. For instance, $aaacc \uplus abbc = aaaabbccc$. Note that ε is the identity

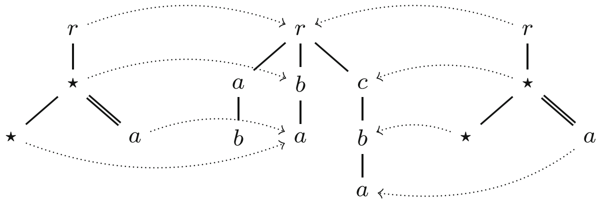


Fig. 3 Embeddings of q_0 in t_0

element of the unordered concatenation $\varepsilon \uplus w = w \uplus \varepsilon = w$ for every unordered word w. Also, given an unordered word w, by w^i we denote the concatenation $w \uplus \ldots \uplus w$ (i times).

A *language* is a set of unordered words. The unordered concatenation of two languages L_1 and L_2 is a language $L_1 \uplus L_2 = \{w_1 \uplus w_2 \mid w_1 \in L_1, w_2 \in L_2\}$. For instance, if $L_1 = \{a, aac\}$ and $L_2 = \{ac, b, \varepsilon\}$, then $L_1 \uplus L_2 = \{a, ab, aac, aabc, aaacc\}$.

Unordered Regular Expressions Analogously to regular expressions, which are used to define languages of ordered words, we propose unordered regular expressions to define languages of unordered words. Essentially, an *unordered regular expression* (URE) defines unordered words by using Kleene star "$*$", disjunction "|", and unordered concatenation "||". Formally, we have the following grammar:

$$E ::= \epsilon \mid a \mid E^* \mid (E\text{“}|\text{”}E) \mid (E\text{“}\|\text{”}E),$$

where $a \in \Sigma$. The semantics of UREs is defined as follows:

$$\begin{aligned}
&L(\epsilon) = \{\varepsilon\},\\
&L(a) = \{a\},\\
&L(E_1 \mid E_2) = L(E_1) \cup L(E_2),\\
&L(E_1 \parallel E_2) = L(E_1) \uplus L(E_2),\\
&L(E^*) = \{w_1 \uplus \ldots \uplus w_i \mid w_1, \ldots, w_i \in L(E) \wedge i \geq 0\}.
\end{aligned}$$

For instance, the URE $(a \parallel (b \mid c))^*$ accepts the unordered words having the number of occurrences of a equal to the total number of b's and c's.

The grammar above uses only one *multiplicity* $*$ and we introduce macros for two other standard and commonly used multiplicities:

$$E^+ := E \parallel E^*, \qquad E^? := E \mid \epsilon.$$

The URE $(a \parallel b^?)^+ \parallel (a \mid c)^?$ accepts the unordered words having at least one a, at most one c, and a number of b's less or equal than the number of a's.

Interval Multiplicities While the multiplicities $*$, $+$, and ? allow to specify unordered words with multiple occurrences of a symbol, we additionally introduce *interval multiplicities* to allow to specify a range of allowed occurrences of a symbol in an unordered word. More precisely, we extend the grammar of UREs by allowing expressions of the form $E^{[n,m]}$ and $E^{[n,m]^?}$, where $n \in \mathbb{N}_0$ and $m \in \mathbb{N}_0 \cup \{\infty\}$. Their semantics is defined as follows:

$$\begin{aligned}
L(E^{[n,m]}) &= \{w_1 \uplus \ldots \uplus w_i \mid w_1, \ldots, w_i \in L(E) \wedge n \leq i \leq m\},\\
L(E^{[n,m]^?}) &= L(E^{[n,m]}) \cup \{\varepsilon\}.
\end{aligned}$$

In the rest of the paper, we write simply *interval* instead of *interval multiplicity*. Furthermore, we view the following standard multiplicities as macros for intervals:

$$* := [0, \infty], \qquad + := [1, \infty], \qquad ? := [0, 1].$$

Additionally, we introduce the single occurrence multiplicity 1 as a macro for the interval $[1, 1]$.

Note that the intervals do not add expressibility to general UREs, but they become useful if we impose some restrictions. For example, if we disallow repetitions of a symbol in a URE and ban expressions of the form $a \parallel a^?$, we can however write $a^{[1,2]}$ to specify a range of occurrences of a symbol in an unordered word without repeating a symbol in the URE.

3 Intractability of Unordered Regular Expressions

In this section, we study the reasons of the intractability of UREs w.r.t. the following two fundamental decision problems: *membership* and *containment*. In Section 3.1 we show that membership is NP-complete even under significant restrictions on the UREs while in Section 3.2 we show that the containment is Π_2^p-hard (and in 3-EXPTIME). We notice that the proofs of both results rely on UREs allowing repetitions of the same symbol. Consequently, we disallow such repetitions and we show that this restriction does not avoid intractability of the containment (Section 3.3). We observe that the proof of this result employs UREs with arbitrary use of disjunction and intervals, and therefore, in Section 4 we impose further restrictions and define the disjunctive interval multiplicity expressions (DIMEs), a subclass for which we show that the two problems of interest become tractable.

3.1 Membership

In this section, we study the problem of deciding the *membership* of an unordered word to the language of a URE. First of all, note that this problem can be easily reduced to testing the membership of a vector to the Parikh image of a regular language, known to be NP-complete [26], and vice versa. We show that deciding the membership of an unordered word to the language a URE remains NP-complete even under significant restrictions on the class of UREs, a result which does not follow from [26].

Theorem 1 *Given an unordered word w and an expression E of the grammar $E ::= a \mid E^? \mid (E \text{“}\parallel\text{”} E)$, deciding whether $w \in L(E)$ is NP-complete.*

Proof To show that this problem is in NP, we point out that a nondeterministic Turing machine guesses a permutation of w and checks whether it is accepted by the NFA corresponding to E with the unordered concatenation replaced by standard concatenation. We recall that w has unary representation.

Next, we prove the NP-hardness by reduction from $\text{SAT}_{\text{1-in-3}}$ i.e., given a 3CNF formula, determine whether there exists a valuation such that each clause has exactly one true literal (and exactly two false literals). The $\text{SAT}_{\text{1-in-3}}$ problem is known to be NP-complete [38]. The reduction works as follows. We take a 3CNF formula $\varphi = c_1 \wedge \ldots \wedge c_k$ over the variables $\{x_1, \ldots, x_n\}$. We take the alphabet $\{d_1, \ldots, d_k, v_1, \ldots, v_n\}$. Each d_i corresponds to a clause c_i (for $1 \leqslant i \leqslant k$) and each

v_j corresponds to a variable x_j (for $1 \leqslant j \leqslant n$). We construct the unordered word $w_\varphi = d_1 \ldots d_k v_1 \ldots v_n$ and the expression $E_\varphi = X_1 \parallel \ldots \parallel X_n$, where for $1 \leqslant j \leqslant n$:

$$X_j = (v_j \parallel d_{t_1} \parallel \ldots \parallel d_{t_l})^? \parallel (v_j \parallel d_{f_1} \parallel \ldots \parallel d_{f_m})^?,$$

and $d_{t_1}, \ldots, d_{t_l}$ (with $1 \leqslant t_1, \ldots, t_l \leqslant k$) correspond to the clauses that use the literal x_j, and $d_{f_1}, \ldots d_{f_m}$ (with $1 \leqslant f_1, \ldots, f_m \leqslant k$) correspond to the clauses that use the literal $\neg x_j$. For example, for the formula $\varphi_0 = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$, we construct $w_{\varphi_0} = d_1 d_2 v_1 v_2 v_3 v_4$ and

$$E_{\varphi_0} = (v_1 \parallel d_1)^? \parallel (v_1 \parallel d_2)^? \parallel v_2^? \parallel (v_2 \parallel d_1)^? \parallel (v_3 \parallel d_1 \parallel d_2)^? \parallel v_3^? \parallel v_4^? \parallel (v_4 \parallel d_2)^?.$$

We claim that $\varphi \in \mathrm{SAT}_{\text{1-in-3}}$ iff $w_\varphi \in L(E_\varphi)$. For the *only if* case, let $V : \{x_1, \ldots, x_n\} \rightarrow \{\mathit{true}, \mathit{false}\}$ be the $\mathrm{SAT}_{\text{1-in-3}}$ valuation of φ. We use V to construct the derivation of w_φ in $L(E_\varphi)$: for $1 \leqslant j \leqslant n$, we take $(v_j \parallel d_{t_1} \parallel \ldots \parallel d_{t_l})$ from X_j if $V(x_j) = \mathit{true}$, and $(v_j \parallel d_{f_1} \parallel \ldots \parallel d_{f_m})$ from X_j otherwise. Since V is a $\mathrm{SAT}_{\text{1-in-3}}$ valuation of φ, each d_i (with $1 \leqslant i \leqslant k$) occurs exactly once, hence $w_\varphi \in L(E_\varphi)$. For the *if* case, we assume that $w_\varphi \in L(E_\varphi)$. Since $w_\varphi(v_j) = 1$, we infer that w_φ uses exactly one of the expressions of the form $(v_j \parallel \ldots)^?$. Moreover, since $w_\varphi(d_i) = 1$, we infer that the valuation encoded in the derivation of w_φ in $L(E_\varphi)$ validates exactly one literal of each clause in φ, and therefore, $\varphi \in \mathrm{SAT}_{\text{1-in-3}}$. Clearly, the described reduction works in polynomial time. □

3.2 Containment

In this section, we study the problem of deciding the *containment* of two UREs. It is well known that regular expression containment is a PSPACE-complete problem [46], but we cannot adapt this result to characterize the complexity of the containment of UREs because the order plays an essential role in the reduction. In this section, we prove that deciding the containment of UREs is Π_2^{P}-hard and we show an upper bound which follows from the complexity of deciding the satisfiability of Presburger logic formulas [36, 44].

Theorem 2 *Given two UREs E_1 and E_2, deciding $L(E_1) \subseteq L(E_2)$ is 1) Π_2^{P}-hard and 2) in 3-EXPTIME.*

Proof 1) We prove the Π_2^{P}-hardness by reduction from the problem of checking the satisfiability of $\forall^*\exists^*$QBF formulas, a classical Π_2^P-complete problem. We take a $\forall^*\exists^*$QBF formula

$$\psi = \forall x_1, \ldots, x_n . \exists y_1, \ldots, y_m . \varphi,$$

where $\varphi = c_1 \wedge \ldots \wedge c_k$ is a quantifier-free CNF formula. We call the variables $x_1, \ldots, x_n$ *universal* and the variables $y_1, \ldots, y_m$ *existential.*

We take the alphabet $\{d_1, \ldots, d_k, t_1, f_1, \ldots, t_n, f_n\}$ and we construct two expressions, E_ψ and E'_ψ. First, $E_\psi = d_1 \parallel \ldots \parallel d_k \parallel X_1 \parallel \ldots \parallel X_n$, where for $1 \leqslant i \leqslant n$ $X_i = ((t_i \parallel d_{a_1} \parallel \ldots \parallel d_{a_l}) \mid (f_i \parallel d_{b_1} \parallel \ldots \parallel d_{b_s}))$, and $d_{a_1}, \ldots d_{a_l}$ (with $1 \leqslant a_1, \ldots, a_l \leqslant k$) correspond to the clauses which use the literal x_i, and $d_{b_1}, \ldots, d_{b_s}$

(with $1 \leqslant b_1, \ldots, b_s \leqslant k$) correspond to the clauses which use the literal $\neg x_i$. For example, for the formula

$$\psi_0 = \forall x_1, x_2. \exists y_1, y_2. (x_1 \vee \neg x_2 \vee y_1) \wedge (\neg x_1 \vee y_1 \vee \neg y_2) \wedge (x_2 \vee \neg y_1),$$

we construct:

$$E_{\psi_0} = d_1 \parallel d_2 \parallel d_3 \parallel ((t_1 \parallel d_1) \mid (f_1 \parallel d_2)) \parallel ((t_2 \parallel d_3) \mid (f_2 \parallel d_1)).$$

Note that there is an one-to-one correspondence between the unordered words in $L(E_\psi)$ and the valuations of the universal variables. For example, given the formula ψ_0, the unordered word $d_1^3 d_2 d_3 t_1 f_2$ corresponds to the valuation V such that $V(x_1) = \mathit{true}$ and $V(x_2) = \mathit{false}$.

Next, we construct $E'_\psi = X_1 \parallel \ldots \parallel X_n \parallel Y_1 \parallel \ldots \parallel Y_m$, where:

- $X_i = ((t_i \parallel d^*_{a_1} \parallel \ldots \parallel d^*_{a_l}) \mid (f_i \parallel d^*_{b_1} \parallel \ldots \parallel d^*_{b_s}))$, and $d_{a_1}, \ldots d_{a_l}$ (with $1 \leqslant a_1, \ldots, a_l \leqslant k$) correspond to the clauses which use the literal x_i, and $d_{b_1}, \ldots, d_{b_s}$ (with $1 \leqslant b_1, \ldots, b_s \leqslant k$) correspond to the clauses which use the literal $\neg x_i$ (for $1 \leq i \leq n$),
- $Y_j = ((d^*_{a_1} \parallel \ldots \parallel d^*_{a_l}) \mid (d^*_{b_1} \parallel \ldots \parallel d^*_{b_s}))$, and $d_{a_1}, \ldots d_{a_l}$ (with $1 \leqslant a_1, \ldots, a_l \leqslant k$) correspond to the clauses which use the literal y_j, and $d_{b_1}, \ldots, d_{b_s}$ (with $1 \leqslant b_1, \ldots, b_s \leqslant k$) correspond to the clauses which use the literal $\neg y_j$ (for $1 \leqslant j \leqslant m$).

For example, for ψ_0 above we construct:

$$E'_{\psi_0} = ((t_1 \parallel d^*_1) \mid (f_1 \parallel d^*_2)) \parallel ((t_2 \parallel d^*_3) \mid (f_2 \parallel d^*_1)) \parallel ((d^*_1 \parallel d^*_2) \mid d^*_3) \parallel (\epsilon \mid d^*_2).$$

We claim that $\models \psi$ iff $E_\psi \subseteq E'_\psi$. For the *only if* case, for each valuation of the universal variables, we take the corresponding unordered word $w \in L(E_\psi)$. Since there exists a valuation of the existential variables which satisfies φ, we use this valuation to construct a derivation of w in $L(E'_\psi)$. For the *if* case, for every unordered word from $L(E_\psi)$, we take its derivation in $L(E'_\psi)$ and we use it to construct a valuation of the existential variables which satisfies φ. Clearly, the described reduction works in polynomial time.

2) The membership of the problem to 3-EXPTIME follows from the complexity of deciding the satisfiability of Presburger logic formulas, which is in 3-EXPTIME [36]. Given two UREs E_1 and E_2, we compute in linear time [44] two existential Presburger formulas for their Parikh images: φ_{E_1} and φ_{E_2}, respectively. Next, we test the satisfiability of the following closed Presburger logic formula: $\forall \overline{x}.\ \varphi_{E_1}(\overline{x}) \Rightarrow \varphi_{E_2}(\overline{x})$. □

While the complexity gap for the containment of UREs (as in Theorem 2) is currently quite important, we believe that this gap may be reduced by working on quantifier elimination for the Presburger formula obtained by translating the containment of UREs (as shown in the second part of the proof of Theorem 2). Although we believe that this problem is Π_2^{P}-complete, its exact complexity remains an open question.

3.3 Disallowing Repetitions

The proofs of Theorem 1 and Theorem 2 rely on UREs allowing repetitions of the same symbol, which might be one of the causes of the intractability. Consequently, from now on we disallow repetitions of the same symbol in a URE. Similar restrictions are commonly used for the regular expressions to maintain practical aspects: *single occurrence regular expressions* (*SOREs*) [7], *conflict-free types* [17, 18, 22], and *duplicate-free DTDs* [33]. While the complexity of the membership of an unordered word to the language of a URE without symbol repetitions has recently been shown to be in PTIME [11], testing containment of two such UREs continues to be intractable.

Theorem 3 *Given two UREs E_1 and E_2 not allowing repetitions of symbols, deciding $L(E_1) \subseteq L(E_2)$ is coNP-hard.*

Proof We show the coNP-hardness by reduction from the complement of 3SAT. Take a 3CNF formula $\varphi = c_1 \wedge \ldots \wedge c_k$ over the variables $\{x_1, \ldots, x_n\}$. We assume w.l.o.g. that each variable occurs at most once in a clause. Take the alphabet $\{a_{ij} \mid 1 \leqslant i \leqslant k, 1 \leqslant j \leqslant n,\ c_i \text{ uses } x_j \text{ or } \neg x_j\}$. We construct the expression $E_\varphi = X_1 \parallel \ldots \parallel X_n$, where $X_j = ((a_{t_1 j} \parallel \ldots \parallel a_{t_l j}) \mid (a_{f_1 j} \parallel \ldots \parallel a_{f_m j})$ (for $1 \leqslant j \leqslant n$), and $c_{t_1}, \ldots, c_{t_l}$ (with $1 \leqslant t_1, \ldots, t_l \leqslant k$) are the clauses which use the literal x_j, and $c_{f_1}, \ldots, c_{f_m}$ (with $1 \leqslant f_1, \ldots, f_m \leqslant k$) are the clauses which use the literal $\neg x_j$. Next, we construct $E'_\varphi = (C_1 \mid \ldots \mid C_k)^{[0,k-1]}$, where $C_i = (a_{ij_1} \mid \ldots \mid a_{ij_p})^+$ (for $1 \leqslant i \leqslant k$), and $x_{j_1}, \ldots, x_{j_p}$ (with $1 \leqslant j_1, \ldots, j_p \leqslant n$) are the variables used by the clause c_i. For example, for

$$\varphi_0 = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4),$$

we obtain:

$$E_{\varphi_0} = (a_{11} \mid a_{21}) \parallel (a_{32} \mid a_{12}) \parallel ((a_{13} \parallel a_{23}) \mid a_{33}) \parallel (\epsilon \mid (a_{24} \parallel a_{34})),$$
$$E'_{\varphi_0} = ((a_{11} \mid a_{12} \mid a_{13})^+ \mid (a_{21} \mid a_{23} \mid a_{24})^+ \mid (a_{32} \mid a_{33} \mid a_{34})^+)^{[0,2]}.$$

Note that there is an one-to-one correspondence between the unordered words w_V in $L(E_\varphi)$ and the valuations V of the variables $x_1, \ldots, x_n$ (*). For example, for above φ_0 and the valuation V such that $V(x_1) = V(x_2) = V(x_3) = \mathit{true}$ and $V(x_4) = \mathit{false}$, the unordered word $w_V = a_{11}a_{32}a_{13}a_{23}a_{24}a_{34}$ is in $L(E_{\varphi_0})$. Moreover, given an $w_V \in L(E_\varphi)$, one can easily obtain the valuation.

We observe that the interval $[0, k-1]$ is used above a disjunction of k expressions of the form C_i and there is no repetition of symbols among the expressions of the form C_i. This allows us to state an instrumental property (**): $w \in L(E_{\varphi'})$ iff there exists an $i \in \{1, \ldots, k\}$ such that none of the symbols used in C_i occurs in w. From (*) and (**), we infer that given a valuation V, $V \models \varphi$ iff $w_V \in L(E_\varphi) \setminus L(E_{\varphi'})$, that yields $\varphi \in$ 3SAT iff $L(E_\varphi) \nsubseteq L(E'_\varphi)$. Clearly, the described reduction works in polynomial time. □

Theorem 3 shows that disallowing repetitions of symbols in a URE does not avoid the intractability of the containment. Additionally, we observe that the proof of Theorem 3 employs UREs with arbitrary use of disjunction and intervals. Consequently, in the next section we impose further restrictions that yield a class of UREs with desirable computational properties.

4 Disjunctive Interval Multiplicity Expressions (DIMEs)

In this section, we present the DIMEs, a subclass of UREs for which membership and containment become tractable. First, we present an intuitive representation of DIMEs with characterizing tuples (Section 4.1). Next, we formally define DIMEs and show that they are precisely captured by their characterizing tuples (Section 4.2). Finally, we use a compact representation of the characterizing tuples to show the tractability of DIMEs (Section 4.3).

4.1 Characterizing Tuples

In this section, we introduce the notion of *characterizing tuple* that is an alternative, more intuitive representation of DIMEs, the subclass of UREs that we formally define in Section 4.2. Recall that by $a \in w$ we denote $w(a) \neq 0$. Given a DIME E, the *characterizing tuple* $\Delta_E = (C_E, N_E, P_E, K_E)$ is as follows.

- The *conflicting pairs of siblings* C_E consisting of all pairs of symbols in Σ such that E defines no word using both symbols simultaneously:

$$C_E = \{(a, b) \in \Sigma \times \Sigma \mid \nexists w \in L(E).\, a \in w \wedge b \in w\}.$$

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

$$N_E = \{(a, w(a)) \in \Sigma \times \mathbb{N}_0 \mid w \in L(E)\}.$$

- The *collections of required symbols* P_E capturing 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:

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

- The *counting dependencies* K_E consisting of pairs of symbols (a, b) such that in every word defined by E, the number of bs is at most the number of as. Note that if both (a, b) and (b, a) belong to K_E, then all unordered words defined by E should have the same number of a's and b's.

$$K_E = \{(a, b) \in \Sigma \times \Sigma \mid \forall w \in L(E).\, w(a) \geqslant w(b)\}.$$

As an example we take $E_0 = a^+ \parallel ((b \parallel c^?)^+ \mid d^{[5,\infty]})$ and we illustrate its characterizing tuple Δ_{E_0}. Because P_E is closed under supersets, we list only its minimal elements:

$$C_{E_0} = \{(b,d), (c,d), (d,b), (d,c)\},$$
$$N_{E_0} = \{(a,i) \mid i \geqslant 1\} \cup \{(b,i) \mid i \geqslant 0\} \cup \{(c,i) \mid i \geqslant 0\} \cup \{(d,i) \mid i = 0 \vee i \geqslant 5\},$$
$$P_{E_0} = \{\{a\}, \{b,d\}, \ldots\},$$
$$K_{E_0} = \{(b,c)\}.$$

We point out that N_E may be infinite and P_E exponential in the size of E. Later on we discuss how to represent both sets in a compact manner while allowing efficient manipulation.

Then, an unordered word w *satisfies* a characterizing tuple Δ_E corresponding to a DIME E, denoted $w \models \Delta_E$, if the following conditions are satisfied:

1. $w \models C_E$ i.e., $\forall (a,b) \in C_E.\ (a \in w \Rightarrow b \notin w) \wedge (b \in w \Rightarrow a \notin w)$,
2. $w \models N_E$ i.e., $\forall a \in \Sigma.\ (a, w(a)) \in N_E$,
3. $w \models P_E$ i.e., $\forall X \in P_E.\ \exists a \in X.\ a \in w$,
4. $w \models K_E$ i.e., $\forall (a,b) \in K_E.\ w(a) \geqslant w(b)$.

For instance, the unordered word *aabbc* satisfies the characterizing tuple Δ_{E_0} corresponding to the aforementioned DIME $E_0 = a^+ \parallel ((b \parallel c^?)^+ \mid d^{[5,\infty]})$ since it satisfies all the four conditions imposed by Δ_{E_0}. On the other hand, note that the following unordered words do not satisfy Δ_{E_0}:

- *abddddd* because it contains at the same time b and d, and $(b,d) \in C_{E_0}$,
- *add* because it has two d's and $(d, 2) \notin N_{E_0}$,
- *aa* because it does not contain any b or d and $\{b,d\} \in P_{E_0}$,
- *abbccc* because it has more c's than b's and $(b,c) \in K_{E_0}$.

In the next section, we define the DIMEs and show that they are precisely captured by characterizing tuples.

4.2 Grammar of DIMEs

An *atom* is $(a_1^{I_1} \parallel \ldots \parallel a_k^{I_k})$, where all I_i's are ? or 1. For example, $(a \parallel b^? \parallel c)$ is an atom, but $(a^{[3,4]} \parallel b)$ is not an atom. A *clause* is $(A_1^{I_1} \mid \ldots \mid A_k^{I_k})$, where all A_i's are atoms and all I_i's are intervals. A clause is *simple* if all I_i's are ? or 1. For example, $(a^{[2,3]} \mid (b^? \parallel c)^*)$ is a clause (which is not simple), $((a^? \parallel b) \mid c^?)$ is a simple clause while $((a^? \parallel b^+) \mid c)$ is not a clause.

A *disjunctive interval multiplicity expression* (DIME) is $(D_1^{I_1} \parallel \ldots \parallel D_k^{I_k})$, where for $1 \leqslant i \leqslant k$ either *1)* D_i is a simple clause and $I_i \in \{+, *\}$, or *2)* D_i is a clause and $I_i \in \{1, ?\}$. Moreover, a symbol can occur at most once in a DIME. For example, $(a \mid (b \parallel c^?)^+) \parallel (d^{[3,4]} \mid e^*)$ is a DIME while $(a \parallel b^?)^+ \parallel (a \mid c)$ is not a DIME because it uses the symbol a twice. A *disjunction-free interval multiplicity expression* (IME) is a DIME which does not use the disjunction operator. An example of IME is $a \parallel (b \parallel c^?)^+ \parallel d^{[3,4]}$. For more practical examples of DIMEs see Examples 3 and 4 from Section 5.

We have tailored DIMEs to be able to capture them with characterizing tuples that permit deciding membership and containment in polynomial time (cf. Section 4.3). As we have already pointed out Section 3.3, a slightly more relaxed restriction on the nesting of disjunction and intervals leads to intractability of the containment (Theorem 3). Even though DIMEs may look very complex, the imposed restrictions are necessary to obtain lower complexity while considering fragments with practical relevance (cf. Section 8).

Next, we show that each DIME can be rewritten as an equivalent *reduced DIME*. Reduced DIMEs may also seem complex, but they are a building block for (i) proving that the language of a DIME is precisely captured by its characterizing tuple (Lemma 1), and (ii) computing the compact representation of the characterizing tuples that yield the tractability of DIMEs (cf. Section 4.3).

Before defining the reduced DIMEs, we need to introduce some additional notations. Given an atom A (resp. a clause D), we denote by Σ_A (resp. Σ_D) the set of symbols occurring in A (resp. D). Given a DIME E, by I_E^a (resp. I_E^A or I_E^D) we denote the interval associated in E to the symbol a (resp. atom A or clause D). Because we consider only expressions without repetitions, this interval is well-defined. Moreover, if E is clear from the context, we write simply I^a (resp. I^A or I^D) instead of I_E^a (resp. I_E^A or I_E^D). Furthermore, given an interval I which can be either $[n, m]$ or $[n, m]^?$, by $I^?$ we understand the interval $[n, m]^?$. In a reduced DIME E, each clause with interval D^I has one of the following three types:

1. $D^I = (A_1 \mid \ldots \mid A_k)^+$, where $k \geqslant 2$ and, for every $i \in \{1, \ldots, k\}$, A_i is an atom such that there exists $a \in \Sigma_{A_i}$ such that $I^a = 1$.

   For example, $((a \parallel b^?) \mid c)^+$ has type 1, but a^+ and $((a^? \parallel b^?) \mid c)^+$ do not.
2. $(A_1^{I_1} \mid \ldots \mid A_k^{I_k})$, where for every $i \in \{1, \ldots, k\}$ *1)* A_i is an atom such that there exists $a \in \Sigma_{A_i}$ such that $I^a = 1$ and *2)* 0 does not belong to the set represented by the interval I_i.

   For example, $(a \mid (b^? \parallel c)^{[5,\infty]})$ and a^+ have type 2, but $(a \mid (b^? \parallel c^?)^{[5,\infty]})$ and $(a^* \mid (b^? \parallel c)^{[5,\infty]})$ do not.
3. $(A_1^{I_1} \mid \ldots \mid A_k^{I_k})$, where for every $i \in \{1, \ldots, k\}$ A_i is an atom and I_i is an interval such that 0 belongs to the set represented by the interval I_i.

   For example, $(a^* \mid (b \parallel c)^{[3,4]^?})$ and $(a^? \parallel b^?)^*$ have type 3, but $(a^? \parallel b^?)^{[3,4]}$ does not.

The reduced DIMEs easily yield the construction of their characterizing tuples. Take a clause with interval D^I from a DIME E and observe that the symbols from Σ_D are present in the characterizing tuple Δ_E as follows.

- If D^I is of type 1, then there is no symbol in Σ_D that occurs in a conflict in C_E. Otherwise, C_E consists of all pairs of distinct symbols (a, b) from Σ_D that appear in different atoms from D^I.
- If D^I is of type 1, then we have $(a, n) \in N_E$ for every $(a, n) \in \Sigma_D \times \mathbb{N}_0$. Otherwise, the possible number of occurrences of every symbol a from Σ_D can be obtained directly from the two intervals above it: the interval of D and the interval of the atom containing a. We explain in Section 4.3 how to precisely construct a compact representation of the potentially infinite set N_E.

- If D^I is of type 1 or 2, then every unordered word defined by E contains at least one of the symbols a from Σ_D having interval $I^a = 1$. More precisely, P_E contains all sets of symbols $X \subseteq \Sigma$ containing, for every atom of D, at least one symbol a with $I^a = 1$. For example, for $((a \parallel b \parallel c^?) \mid (d \parallel e))^+$, P_E consists of the sets $\{a, d\}$, $\{a, e\}$, $\{b, d\}$, $\{b, e\}$ and all their supersets. Otherwise, if D^I is of type 3, then there is no set in P_E containing only symbols from Σ_D.
- Regardless of the type of D^I, the counting dependencies K_E consist of all pairs of symbols (a, b) such that they appear in the same atom in D and $I^a = 1$.

To obtain reduced DIMEs, we use the following rules:

- Take a simple clause $(A_1^{I_1} \mid \ldots \mid A_k^{I_k})$.
  - $(A_1^{I_1} \mid \ldots \mid A_k^{I_k})^*$ goes to $A_1^* \parallel \ldots \parallel A_k^*$ (k clauses of type 3). Essentially, we distribute the $*$ of a disjunction of atoms with intervals to each of the atoms. For example, $(a \mid (b \parallel c^?))^*$ goes to $a^* \parallel (b \parallel c^?)^*$.
  - $(A_1^{I_1} \mid \ldots \mid A_k^{I_k})^+$ goes to $A_1^* \| \ldots \| A_k^*$ (k clauses of type 3) if there exists an atom with interval $A_i^{I_i}$ ($i \in \{1, \ldots, k\}$) that defines the empty word i.e., $I_i =?$ or $I^a =?$ for every symbol $a \in \Sigma_{A_i}$. If the empty word is defined, then we can basically transform the $+$ into $*$ and then distribute the $*$ as for the previous case. For example, $((a \parallel b^?) \mid (c \parallel d)^?)^+$ goes to $(a \parallel b^?)^* \parallel (c \parallel d)^*$.
- Take a clause $(A_1^{I_1} \mid \ldots \mid A_k^{I_k})$.
  - $(A_1^{I_1} \mid \ldots \mid A_k^{I_k})^?$ goes to $(A_1^{I_1^?} \mid \ldots \mid A_k^{I_k^?})$ (type 3). We essentially distribute the ? of a disjunction of atoms with intervals to each of the atoms. For example, $(a^{[2,3]} \mid b^+)^?$ goes to $(a^{[2,3]^?} \mid b^*)$.
  - $(A_1^{I_1} \mid \ldots \mid A_k^{I_k})$ goes to $(A_1^{I_1^?} \mid \ldots \mid A_k^{I_k^?})$ (type 3) if there exists an atom with interval $A_i^{I_i}$ ($i \in \{1, \ldots, k\}$) that defines the empty word i.e., 0 belongs to the set represented by I_i or $I^a =?$ for every symbol $a \in \Sigma_{A_i}$. If the empty word is defined by one of the atoms, then we can basically distribute ? to all of them. For example, $(a \mid (b \parallel c)^{[0,5]})$ goes to $(a^? \mid (b \parallel c)^{[0,5]})$.
- Take an atom $(a_1^? \parallel \ldots \parallel a_k^?)$ and an interval I. Then, $(a_1^? \parallel \ldots \parallel a_k^?)^I$ goes to $(a_1^? \parallel \ldots \parallel a_k^?)^{[0,\max(I)]}$, where by $\max(I)$ we denote the maximum value from the set represented by the interval I. This step may be combined with one of the previous ones to rewrite a clause with interval as one of type 3. For example, $((a^? \parallel b^?)^{[3,6]} \mid c)$ goes to $((a^? \parallel b^?)^{[0,6]} \mid c^?)$.
- Remove symbols a (resp. atoms A or clauses D) such that I^a (resp. I^A or I^D) is $[0, 0]$.

Note that each of the rewriting steps gives an equivalent reduced expression.

Next, we assume that we work with reduced DIMEs only and show that the language defined by a DIME E comprises of all unordered words satisfying the characterizing tuple Δ_E.

Lemma 1 *Given an unordered word w and a DIME E, $w \in L(E)$ iff $w \models \Delta_E$.*

Proof The *only if* part follows from the definition of the satisfiability of Δ_E. For the *if* part, we take the tuple Δ_E corresponding to a DIME $E = D_1^{I_1} \parallel \ldots \parallel D_k^{I_k}$ and an unordered word w such that $w \models \Delta_E$. Let $w = w_1 \uplus \ldots \uplus w_k \uplus w'$, where each w_i contains all occurrences in w of the symbols from Σ_{D_i} (for $1 \leqslant i \leqslant k$). Since $w \models N_E$, we infer that there is no symbol $a \in \Sigma \setminus (\Sigma_{D_1} \cup \ldots \cup \Sigma_{D_k})$ such that $a \in w$, which implies $w' = \varepsilon$. Thus, proving $w \models E$ reduces to proving that $w_i \models D_i^{I_i}$ (for $1 \leqslant i \leqslant k$). Since E is a reduced DIME, each derivation can be constructed by reasoning on the three possible types of the $D_i^{I_i}$ (for $1 \leqslant i \leqslant k$).

Case 1 Take $D_i^{I_i} = (A_1 \mid \ldots \mid A_k)^+$ of type 1. From the semantics of the UREs, we observe that proving $w_i \models D_i^{I_i}$ is equivalent to proving that (i) w_i is non-empty and (ii) w_i can be split as $w_i = w_1' \uplus \ldots \uplus w_p'$, where every w_j' $(1 \leqslant j \leqslant p)$ satisfies an atom A_l $(1 \leqslant l \leqslant k)$. First, we point out that since w satisfies the collections of required symbols P_E, we infer that w_i is non-empty, which implies (i). Then, since w satisfies the extended cardinality map N_E and the counting dependencies K_E, we infer that (ii) is also satisfied.

Case 2 Take $D_i^{I_i} = (A_1^{I_1} \mid \ldots \mid A_k^{I_k})$ of type 2. From the semantics of UREs, we observe that proving $w_i \models D_i^{I_i}$ is equivalent to proving that (i) w_i is non-empty and (ii) there exists an atom with interval $A_j^{I_j}$ $(1 \leqslant j \leqslant k)$ such that $w_i \models A_j^{I_j}$. Since $w \models P_E$, we infer that w_i is non-empty hence (i) is satisfied. Then, since $w \models C_E$, we infer that only the symbols from one atom A_j of D_i are present in w_i. Moreover, since $w \models N_E$ and $w \models K_E$, we infer that the number of occurrences of each symbol from Σ_{A_j} are such that $w_i \models A_j^{I_j}$. Hence, the condition (ii) is also satisfied.

Case 3 Take $D_i^{I_i} = (A_1^{I_1} \mid \ldots \mid A_k^{I_k})$ of type 3. The only difference w.r.t the previous case is that w_i may be also empty, hence proving $w_i \models D_i^{I_i}$ is equivalent to proving only that there exists an atom with interval $A_j^{I_j}$ $(1 \leqslant j \leqslant k)$ such that $w_i \models A_j^{I_j}$, which follows similarly to the previous case. □

Moreover, we define the *subsumption* of two characterizing tuples, which captures the containment of DIMEs. Given two DIMEs E and E', we write $\Delta_{E'} \preccurlyeq \Delta_E$ if $C_E \subseteq C_{E'}$, $N_{E'} \subseteq N_E$, $P_E \subseteq P_{E'}$, and $K_E \subseteq K_{E'}$. Then, we obtain the following.

Lemma 2 *Given two DIMEs E and E', $L(E') \subseteq L(E)$ iff $\Delta_{E'} \preccurlyeq \Delta_E$.*

Proof First, we claim that given two DIMEs E and E': $\Delta_{E'} \preccurlyeq \Delta_E$ iff $w \models \Delta_{E'}$ implies $w \models \Delta_E$ for every w (*). The *only if* part of (*) follows directly from the definitions while the *if* part can be easily shown by contraposition. From Lemma 1 and (*) we infer the correctness of Lemma 2. □

Example 1 For the following DIMEs, it holds that $L(E') \subsetneq L(E)$ and $L(E) \not\subseteq L(E')$:

- Take $E = a^* \,\|\, b^*$ and $E' = (a \,\|\, b^?)^*$. Note that $K_E = \varnothing$ and $K_{E'} = \{(a, b)\}$. For instance, the unordered word b belongs to $L(E)$, but does not belong to $L(E')$.
- Take $E = a^{[3,6]^?} \mid b^*$ and $E' = a^{[3,6]} \mid b^+$. Note that $P_E = \varnothing$, and $P_{E'} = \{\{a, b\}\}$. For instance, the unordered word ε belongs to $L(E)$, but does not belong to $L(E')$.
- Take $E = (a \,\|\, b^?)^*$ and $E' = (a \,\|\, b^?)^{[0,5]}$. Note that $(a, 6)$ belongs to N_E, but not to $N_{E'}$. For instance, the unordered word a^6 belongs to $L(E)$, but does not belong to $L(E')$.
- Take $E = (a \mid b)^+$ and $E' = a^+ \mid b^+$. Note that $C_E = \varnothing$, and $C_{E'} = \{(a, b), (b, a)\}$. For instance, the unordered word ab belongs to $L(E)$, but does not belong to $L(E')$. □

Lemma 2 shows that two equivalent DIMEs yield the same characterizing tuple, and hence, the tuple Δ_E can be viewed as a "canonical form" for the language defined by a DIME E. Formally, we obtain the following.

Corollary 1 *Given two DIMEs E and E', $L(E) = L(E')$ iff $\Delta_E = \Delta_{E'}$.*

In the next section, we show that the characterizing tuple has a compact representation that permits us to decide the problems of membership and containment in polynomial time.

4.3 Tractability of DIMEs

We now show that the characterizing tuple admits a compact representation that yields the tractability of deciding membership and containment of DIMEs.

Given a reduced DIME E, note that C_E and K_E are quadratic in $|\Sigma|$ and can be easily constructed. The set C_E consists of all pairs of distinct symbols (a, b) such that they appear in different atoms in the same clause of type 2 or 3. Moreover, K_E consists of all pairs of distinct symbols (a, b) such that they appear in the same atom and $I^a = 1$.

While N_E may be infinite, it can be easily represented in a compact manner using intervals: for every symbol a, the set $\{i \in \mathbb{N}_0 \mid (a, i) \in N_E\}$ is representable by an interval. Given a symbol $a \in \Sigma$, by $\hat{N}_E(a)$ we denote the interval representing the set $\{i \in \mathbb{N}_0 \mid (a, i) \in N_E\}$ that can be easily obtained from E:

- $\hat{N}_E(a) = [0, 0]$ if a appears in no clause in E,
- $\hat{N}_E(a) = [0, \infty]$ (or simply $*$) if a appears in a clause of type 1 in E,
- $\hat{N}_E(a) = I^A$ if $I^a = 1$, A is the atom containing a, and A is the unique atom of a clause of type 2 or 3,
- $\hat{N}_E(a) = I^{A^?}$ if $I^a = 1$, A is the atom containing a, and A appears in a clause of type 2 or 3 containing at least two atoms,
- $\hat{N}_E(a) = [0, \max(I^A)]$ if $I^a =?$, A is the atom containing a, and A appears in a clause of type 2 or 3.

For example, for $E_0 = a^+ \parallel ((b \parallel c^?)^+ \mid d^{[5,\infty]})$, we obtain the following $\hat{N}_{E_0}$:

$$\hat{N}_{E_0}(a) = +, \qquad \hat{N}_{E_0}(b) = *, \qquad \hat{N}_{E_0}(c) = *, \qquad \hat{N}_{E_0}(d) = [5, \infty]^?.$$

Naturally, testing $N_{E'} \subseteq N_E$ reduces to a simple test on $\hat{N}_{E'}$ and $\hat{N}_E$.

Representing P_E in a compact manner is more tricky. A natural idea would be to store only its $\subseteq$-minimal elements since P_E is closed under supersets. Unfortunately, there exist DIMEs having an exponential number of $\subseteq$-minimal elements. For instance, for the DIME $E_1 = ((a \parallel b) \mid (c \parallel d))^+ \parallel ((e \parallel f)^{[2,5]} \mid g^{[1,3]}) \parallel (h^* \parallel i^{[0,9]})$, the set P_{E_1} has 6 $\subseteq$-minimal elements: $\{a, c\}$, $\{a, d\}$, $\{b, c\}$, $\{b, d\}$, $\{e, g\}$, and $\{f, g\}$. The example easily generalizes to arbitrary numbers of atoms used in the clauses.

However, we observe that the exponentially-many $\subseteq$-minimal elements may contain redundant information that is already captured by other elements of the characterizing tuple. For instance, for the above DIME E_1, if we know that $\{a, c\}$ belongs to P_E, we can easily see that other $\subseteq$-minimal elements also belong to P_E. More precisely, we observe that for every unordered word w defined by E it holds that $w(a) = w(b)$, $w(c) = w(d)$ and $w(e) = w(f)$, which is captured by the counting dependencies $K_E = \{(a, b), (b, a), (c, d), (d, c), (e, f), (f, e)\}$. Hence, for the unordered words defined by E, the presence of an a implies the presence of a b, the presence of a c implies the presence of a d, etc. Consequently, if $\{a, c\}$ belongs to P_E, then $\{b, c\}$, $\{a, d\}$, and $\{b, d\}$ also belong to P_E. Similarly, if $\{e, g\}$ belongs to P_E, then $\{f, g\}$ also belongs to P_E.

Next, we use the aforementioned observation to define a compact representation of P_E. For this purpose, we introduce the auxiliary notion of symbols *implied by a DIME E in the presence of a set of symbols X*, denoted $impl_E(X)$:

$$impl_E(X) = X \cup \{a \in \Sigma \mid \exists b \in X.\ (a, b) \in K_E \text{ and } (b, a) \in K_E\}.$$

For example, for the above E_1, we have $impl_E(\{a, c\}) = \{a, b, c, d\}$.

Moreover, given a DIME E, by $P_E^{\subseteq\min}$ we denote the set of all $\subseteq$-minimal elements of P_E. Given a subset $P \subseteq P_E^{\subseteq\min}$, we say that P is:

- *non-redundant* if $\forall X \in P.\ \nexists Y \in P.\ X \subseteq impl_E(Y)$,
- *covering* if $\forall X \in P_E^{\subseteq\min}.\ \exists Y \in P.\ X \subseteq impl_E(Y)$.

For example, take the above $E_1 = ((a\|b) \mid (c\|d))^+\|((e\|f)^{[2,5]} \mid g^{[1,3]})\|(h^*\|i^{[0,9]})$ and recall that $P_{E_1}^{\subseteq\min} = \{\{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{e, g\}, \{f, g\}\}$. Then, we have the following:

- $\{\{b, c\}, \{f, g\}\}$ is non-redundant and covering,
- $\{\{b, c\}\}$ is non-redundant and it is not covering,
- $\{\{a, c\}, \{b, c\}, \{f, g\}\}$ is redundant and covering,
- $\{\{a, c\}, \{b, c\}\}$ is redundant and not covering.

Given a DIME E, the *compact representation of the collections of required symbols* P_E is naturally a non-redundant and covering subset of $P_E^{\subseteq\min}$. Since there may exist many non-redundant and covering subsets of $P_E^{\subseteq\min}$, we use the total order $<_\Sigma$ on the alphabet Σ to propose a deterministic construction of the compact representation $\hat{P}_E$. For this purpose, we define first some additional notations.

Given an atom A, by $\Phi(A)$ we denote the smallest label from Σ w.r.t. $<_\Sigma$ that is present in A and has interval 1:

$$\Phi(A) = \min_{<_\Sigma}\{a \in \Sigma_A \mid I^a = 1\}.$$

For example, $\Phi(a \parallel b) = a$. Then, given a clause with interval D^I, by $\Phi(D^I)$ we denote the set of all symbols $\Phi(A)$ for every atom A in D:

$$\Phi(D^I) = \{\Phi(A) \mid A \text{ is an atom in } D\}.$$

For example, $\Phi(((a \parallel b) \mid (c \parallel d))^+) = \{a, c\}$ and $\Phi(((e \parallel f)^{[2,5]} \mid g^{[1,3]})) = \{e, g\}$. Then, $\hat{P}(E)$ consists of all such sets for the clauses with intervals of type 1 or 2:

$$\hat{P}_E = \{\Phi(D^I) \mid D^I \text{ is a clause with interval of type 1 or 2 in } E\}.$$

For example, $\hat{P}_{E_1} = \{\{a, c\}, \{e, g\}\}$. Notice that the set $\{a, c\}$ is due to the clause with interval $((a \parallel b) \mid (c \parallel d))^+$ of type 1 and the set $\{e, g\}$ is due to the clause with interval $((e \parallel f)^{[2,5]} \mid g^{[1,3]})$ of type 2. Also notice that the clause with interval $(h^* \parallel i^{[0,9]})$ is of type 3, none of its symbols is required, and consequently, no set in $\hat{P}_E$ contains symbols from it.

We have introduced all elements to be able to define the compact representation of a characterizing tuple. Given a DIME E, we say that $\hat{\Delta} = (C_E, \hat{N}_E, \hat{P}_E, K_E)$ is the *compact representation* of its characterizing tuple Δ_E. Then, an unordered word w satisfies $\hat{\Delta}_E$, denoted $w \models \hat{\Delta}_E$, if

- $w \models C_E$ and $w \models K_E$ as previously defined when we have introduced $w \models \Delta_E$,
- $w \models \hat{N}_E$ i.e., $\forall a \in \Sigma.\ w(a) \in \hat{N}_E(a)$,
- $w \models \hat{P}_E$ i.e., $\forall X \in \hat{P}_E.\ \exists a \in X.\ a \in w$. Notice that we use exactly the same definition as for $w \models P_E$ and recall that $\hat{P}_E$ is in fact a non-redundant and covering subset of $P_E^{\subseteq\min}$.

Next, we show that given a DIME E, its compact characterizing tuple $\hat{\Delta}_E$ defines precisely the same set of unordered words as its characterizing tuple Δ_E.

Lemma 3 *Given an unordered word w and a DIME E, $w \models \Delta_E$ iff $w \models \hat{\Delta}_E$.*

Proof The *only if* part follows directly from the definitions. For the *if* part, proving $w \models \Delta_E$ reduces to proving that $w \models P_E$, which moreover, reduces to proving that for every X from $P_E^{\subseteq\min}$ there is a symbol a in X that occurs in w (*). Since $\hat{P}_E$ is a covering subset of $P_E^{\subseteq\min}$, we know that for every $X \in P_E^{\subseteq\min}$ there exists a set $Y \in \hat{P}_E$ such that $X \subseteq \mathit{impl}_E(Y)$. Since $w \models \hat{P}_E$ and $w \models K_E$, we infer that (*) is satisfied. □

Additionally, we define the *subsumption* of the compact representations of two characterizing tuples. Given two DIMEs E and E', we write $\hat{\Delta}_{E'} \preccurlyeq \hat{\Delta}_E$ if

- $C_E \subseteq C_{E'}$ and $K_E \subseteq K_{E'}$ (as for the subsumption of characterizing tuples),
- $\forall a \in \Sigma.\ \hat{N}_{E'}(a) \subseteq \hat{N}_E(a)$,
- $\forall X \in \hat{P}_E.\ \exists Y \in \hat{P}_{E'}.\ Y \subseteq \mathit{impl}_{E'}(X)$.

Next, we show that the subsumption of compact representations of characterizing tuples captures the subsumption of characterizing tuples.

Lemma 4 *Given two DIMEs E and E', $\Delta_{E'} \preccurlyeq \Delta_E$ iff $\hat{\Delta}_{E'} \preccurlyeq \hat{\Delta}_E$.*

Proof First, since P_E is closed under supersets, we observe that

$$P_E \subseteq P_{E'} \text{ iff } \forall X \in P_E^{\subseteq\min}.\ \exists Y \in P_{E'}^{\subseteq\min}.\ Y \subseteq X.$$

Moreover, the conditions $C_E \subseteq C_{E'}$ and $K_E \subseteq K_{E'}$ are part of both $\Delta_{E'} \preccurlyeq \Delta_E$ and $\hat{\Delta}_{E'} \preccurlyeq \hat{\Delta}_E$. Consequently, proving $\Delta_{E'} \preccurlyeq \Delta_E$ iff $\hat{\Delta}_{E'} \preccurlyeq \hat{\Delta}_E$ reduces to proving that, if $C_E \subseteq C_{E'}$ and $K_E \subseteq K_{E'}$, then

$$\forall X \in P_E^{\subseteq\min}.\ \exists Y \in P_{E'}^{\subseteq\min}.\ Y \subseteq X \text{ iff } \forall X \in \hat{P}_E.\ \exists Y \in \hat{P}_{E'}.\ Y \subseteq \mathit{impl}_{E'}(X).$$

For the *only if* part, take a set X from $\hat{P}_E$. Since X also belongs to $P_E^{\subseteq\min}$, we know by hypothesis that there exists a set Y in $P_{E'}^{\subseteq\min}$ such that $Y \subseteq X$. Then, construct a set Y' from Y by replacing each symbol b from Y with the smallest a w.r.t. $<_\Sigma$ such that (a, b) and (b, a) belong to $K_{E'}$. Moreover, since $K_E \subseteq K_{E'}$, we infer that $Y' \subseteq \mathit{impl}_{E'}(X)$. For the *if* part, take an X from $\hat{P}_E$ and an Y from $\hat{P}_{E'}$ s.t. $Y \subseteq \mathit{impl}_{E'}(X)$. To construct the corresponding X' in $P_E^{\subseteq\min}$ and Y' in $P_{E'}^{\subseteq\min}$ such that $Y' \subseteq X'$, we replace symbols a from X and a' from Y with symbols b in X' and b' in Y' such that (a, b) and (b, a) belong to K_E, and (a', b') and (b', a') belong to $K_{E'}$. Since $K_E \subseteq K_{E'}$, we know that such X' and Y' do exist. □

Example 2 Take $E = a^* \parallel (b \mid c)^+ \parallel d^*$ and $E' = (a \parallel b)^+ \mid (c \parallel d)^+$. Notice that $L(E') \subseteq L(E)$, $\Delta_{E'} \preccurlyeq \Delta_E$, and $\hat{\Delta}_{E'} \preccurlyeq \hat{\Delta}_E$. In particular, we have the following.

- $C_E = \varnothing$ is included in $C_{E'} = \{(a, c), (a, d), (b, c), (b, d), (c, a), (c, b), (d, a), (d, b)\}$,
- $\hat{N}_E(a) = \hat{N}_{E'}(a) = *, \ldots, \hat{N}_E(d) = \hat{N}_{E'}(d) = *$,
- $K_E = \varnothing$ is included in $K_{E'} = \{(a, b), (b, a), (c, d), (d, c)\}$,
- $\hat{P}_E = \{\{b, c\}\}$ and $\hat{P}_{E'} = \{\{a, c\}\}$ that compactly represent $P_E = \{\{b, c\}, \ldots\}$ and $P_{E'} = \{\{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \ldots\}$, respectively (we have listed only the $\subseteq$-minimal sets). Then, take $X = \{b, c\}$ from $\hat{P}_E$ and notice that there exists $Y = \{a, c\}$ in $\hat{P}_{E'}$ such that $Y \subseteq \mathit{impl}_{E'}(X)$ because $\mathit{impl}_{E'}(\{b, c\}) = \{a, b, c, d\}$. □

Next, we show that the compact representation is of polynomial size.

Lemma 5 *Given a DIME E, the compact representation $\hat{\Delta}_E = (C_E, \hat{N}_E, \hat{P}_E, K_E)$ of its characterizing tuple Δ_E is of size polynomial in the size of the alphabet Σ.*

Proof By construction, the sizes of C_E and K_E are quadratic in $|\Sigma|$ while the sizes of $\hat{P}_E$ and $\hat{N}_E$ are linear in $|\Sigma|$. □

The use of compact representation of characterizing tuples allows us to state the main result of this section.

Theorem 4 *Given an unordered word w and two DIMEs E and E':*

1. *deciding whether $w \in L(E)$ is in PTIME,*
2. *deciding whether $L(E') \subseteq L(E)$ is in PTIME.*

Proof The first part follows from Lemma 1, Lemma 3, and Lemma 5. The second part follows from Lemma 2, Lemma 4, and Lemma 5. □

5 Interval Multiplicity Schemas

In this section, we employ DIMEs to define schema languages and we present the related problems of interest.

Definition 1 A *disjunctive interval multiplicity schema (DIMS)* is a tuple $S = (root_S, R_S)$, where $root_S \in \Sigma$ is a designated root label and R_S maps symbols in Σ to DIMEs. By *DIMS* we denote the set of all disjunctive interval multiplicity schemas. A *disjunction-free interval multiplicity schema (IMS)* $S = (root_S, R_S)$ is a restricted DIMS, where R_S maps symbols in Σ to IMEs. By *IMS* we denote the set of all disjunction-free interval multiplicity schemas.

We define the language captured by a DIMS S in the following way. Given a tree t, we first define the unordered word ch_t^n of children of a node $n \in N_t$ of t i.e., $ch_t^n(a) = |\{m \in N_t \mid (n, m) \in child_t \wedge lab_t(m) = a\}|$. Now, a tree t *satisfies* S, in symbols $t \models S$, if $lab_t(root_t) = root_S$ and for every node $n \in N_t$, $ch_t^n \in L(R_S(lab_t(n)))$. By $L(S) \subseteq Tree$ we denote the set of all trees satisfying S.

In the sequel, we present a schema $S = (root_S, R_S)$ as a set of rules of the form $a \to R_S(a)$, for every $a \in \Sigma$. If $L(R_S(a)) = \varepsilon$, then we write $a \to \epsilon$ or we simply omit writing such a rule.

Example 3 Take the content model of a semi-structured database storing information about a peer-to-peer file sharing system, having the following rules: *1)* a peer is allowed to download at most the same number of files that it uploads, and *2)* peers are split into two groups: a peer is a *vip* if it uploads at least 100 files, otherwise it is a simple *user*:

$$\begin{aligned} peers \to \quad & user^* \parallel vip^*, \\ user \to \quad & (upload \parallel download^?)^{[0,99]}, \\ vip \to \quad & (upload \parallel download^?)^{[100,\infty]}. \end{aligned}$$

□

Example 4 Take the content model of a semi-structured database storing information about two types of cultural events: plays and movies. Every event has a date when it takes place. If the event is a play, then it takes place in a theater while a movie takes place in a cinema.

$$\begin{aligned} events \to \quad & event^*, \\ event \to \quad & date \parallel ((play \parallel theater) \mid (movie \parallel cinema)). \end{aligned}$$

□

Problems of Interest We define next the problems of interest and we formally state the corresponding decision problems parameterized by the class of schema $\mathcal{S}$ and, when appropriate, by a class of queries $\mathcal{Q}$.

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

$$\mathrm{SAT}_{\mathcal{S}} = \{S \in \mathcal{S} \mid \exists t \in \mathit{Tree}.\ t \models S\}.$$

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

$$\mathrm{MEMB}_{\mathcal{S}} = \{(S, t) \in \mathcal{S} \times \mathit{Tree} \mid t \models S\}.$$

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

$$\mathrm{CNT}_{\mathcal{S}} = \{(S_1, S_2) \in \mathcal{S} \times \mathcal{S} \mid L(S_1) \subseteq L(S_2)\}.$$

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

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

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

$$\mathrm{IMPL}_{\mathcal{S},\mathcal{Q}} = \{(S, q) \in \mathcal{S} \times \mathcal{Q} \mid \forall t \in L(S).\ t \models q\}.$$

- *Query containment in the presence of schema* – checking if every tree satisfying the given schema and one given query also satisfies another given query:

$$\mathrm{CNT}_{\mathcal{S},\mathcal{Q}} = \{(p, q, S) \in \mathcal{Q} \times \mathcal{Q} \times \mathcal{S} \mid \forall t \in L(S).\ t \models p \Rightarrow t \models q\}.$$

We study these problems for DIMSs and IMSs in Sections 6 and 7 of the paper.

6 Complexity of Disjunctive Interval Multiplicity Schemas (DIMSs)

In this section, we present the complexity results for DIMSs. First, we show the tractability of schema satisfiability and containment. Then, we provide an algorithm for deciding membership in *streaming* i.e., that processes an XML document in a single pass and using memory depending on the height of the tree and not on its size. Finally, we point out that the complexity of query satisfiability, implication, and containment in the presence of the schema follow from existing results.

First, we show the tractability of schema satisfiability and schema containment.

Proposition 1 $\mathrm{SAT}_{\mathit{DIMS}}$ *and* $\mathrm{CNT}_{\mathit{DIMS}}$ *are in PTIME.*

Proof A simple algorithm based on dynamic programming can decide the satisfiability of a DIMS. More precisely, given a schema $S = (\mathit{root}_S, R_S)$, one has to determine for every symbol a of the alphabet Σ whether there exists a (finite) tree t that satisfies $S' = (a, R_S)$. Then, the schema S is satisfiable if there exist such a tree for the root label root_S.

Moreover, testing the containment of two DIMSs reduces to testing, for each symbol in the alphabet, the containment of the associated DIMEs, which is in PTIME (Theorem 4). □

Next, we provide an algorithm for deciding membership in *streaming* i.e., that processes an XML document in a single pass and uses memory depending on the height of the tree and not on its size. Our notion of streaming has been employed in [42] as a relaxation of the constant-memory XML validation against DTDs, which can be performed only for some DTDs [41, 42]. In general, validation against DIMSs cannot be performed with constant memory due to the same observations as in [41, 42] w.r.t. the use of recursion in the schema. Hence, we have chosen our notion of streaming to be able to have an algorithm that works for the entire class of DIMSs. We assume that the input tree is given in XML format, with arbitrary ordering of sibling nodes. Moreover, the proposed algorithm has *earliest rejection* i.e., if the given tree does not satisfy the given schema, the algorithm outputs the result as early as possible. For a tree t, $height(t)$ 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 Σ.

Proposition 2 MEMB$_{DIMS}$ *is in PTIME. There exists an earliest rejection streaming algorithm that checks membership of a tree t in a DIMS S in time $O(|t| \times |\Sigma|^2)$ and using space $O(height(t) \times |\Sigma|^2)$.*

Proof We propose Algorithm 1 for deciding the membership of a tree t to the language of a DIMS S. The input tree t is given in XML format, with some arbitrary ordering of sibling nodes. We assume a well-formed stream $\widetilde{t} \subset \{open, close\} \times \Sigma$ representing a tree t and a procedure $read(\widetilde{t})$ that returns the next pair (θ, b) in the stream, where $\theta \in \{open, close\}$ and $b \in \Sigma$. The algorithm works for every arbitrary ordering of sibling nodes. To validate a tree t against a DIMS $S = (root_S, R_S)$, one has to run Algorithm 1 after reading the opening tag of the root.

For a given node, the algorithm constructs the compact representation of the characterizing tuple of its label (line 1), which requires space $O(|\Sigma|^2)$ (cf. Lemma 5). The algorithm also stores for a given node the number of occurrences of each label in Σ among its children. This is done using the array *count*, which requires space $O(\Sigma)$. Initially, all values in the array *count* are set at 0 (lines 2-3) and they are updated after reading the open tag of the children (lines 4-6). During the execution, the algorithm maintains a stack whose height is the depth of the currently visited node. Naturally, the bound on space required is $O(height(t) \times |\Sigma|^2)$.

The algorithm has earliest rejection since it rejects a tree as early as possible. More precisely, this can be done after reading the opening tag for nodes that violate the maximum value for the allowed cardinality for their label (lines 7-8) or violate some conflicting pair of siblings (lines 9-10). If it is not the case, the algorithm recursively validates the corresponding subtree (lines 11-12). After reading all children of the current node, the algorithm checks whether the components of the characterizing tuple are satisfied: the extended cardinality map (lines 14-15), the collections of required symbols (lines 16-17), and the counting dependencies (lines 18-19). Notice that since we have checked the conflicting pairs of siblings after reading each opening tag, we do not need to check them again after reading all children. However, we still need to check the extended cardinality map at this moment to see whether the number of occurrences of each label is in the allowed interval. When we have read the opening tag, we were able to reject only if the maximum value for the allowed number of occurrences has been already violated. As for the collections of required symbols and the counting dependencies, we are able to establish whether they are satisfied or not after reading all children. If none of the constraints imposed by the characterizing tuple is violated, the algorithm returns true (line 20). As we have already shown with Lemma 1 and Lemma 3, the compact representation of the characterizing tuple captures precisely the language of a given DIME. Consequently, the algorithm returns true after reading the root node iff the given tree satisfies the given schema. □

Algorithm 1 Streaming algorithm for testing membership.

```
algorithm $validate(a)$
Parameters: DIMS $S$, stream $\widetilde{t}$
Input: the label $a \in \Sigma$ of the current node
Output: $true$ if the subtree rooted at the current node is valid w.r.t. $S$, $false$ otherwise
 1: let $(C, \hat{N}, \hat{P}, K)$ be the compact representation of the characterizing tuple of $R_S(a)$
 2: for $b \in \Sigma$ do
 3:     let $count[b] = 0$
 4: $(\theta, b) = read(\widetilde{t})$
 5: while $\theta = open$ do
 6:     $count[b] := count[b] + 1$
 7:     if $count[b] > \max(\hat{N}(b))$ then
 8:         return $false$
 9:     if $\exists c \in \Sigma.\ (b, c) \in C \wedge count[c] \neq 0$ then
10:         return $false$
11:     if $validate(b) = false$ then
12:         return $false$
13:     $(\theta, b) = read(\widetilde{t})$
14: if $\exists b \in \Sigma.\ count[b] \notin \hat{N}(b)$ then
15:     return $false$
16: if $\exists X \in \hat{P}.\ \forall b \in X.\ count[b] = 0$ then
17:     return $false$
18: if $\exists (b, c) \in K.\ count[b] < count[c]$ then
19:     return $false$
20: return $true$
```

We continue with complexity results that follow from known facts. Query satisfiability for DTDs is NP-complete [5] and we adapt the result for DIMSs.

Proposition 3 SAT$_{DIMS,Twig}$ *is NP-complete.*

Proof Proposition 4.2.1 from [5] implies that satisfiability of twig queries in the presence of DTDs is NP-hard. We adapt the proof and we obtain the following reduction from SAT to SAT$_{DIMS,Twig}$: we take a CNF formula $\varphi = \bigwedge_{i=1}^{n} C_i$ over the variables $x_1, \ldots, x_m$, where each C_i is a disjunction of literals. We take $\Sigma = \{r, t_1, f_1, \ldots, t_m, f_m, c_1, \ldots, c_n\}$ and we construct:

- The DIMS S having the root label r and the rules:
  - $r \to (t_1 \mid f_1) \parallel \ldots \parallel (t_m \mid f_m)$,
  - $t_j \to c_{j_1} \parallel \ldots \parallel c_{j_k}$, where $c_{j_1}, \ldots, c_{j_k}$ correspond to the clauses using x_j (for $1 \leqslant j \leqslant m$),
  - $f_i \to c_{j_1} \parallel \ldots \parallel c_{j_k}$, where $c_{j_1}, \ldots, c_{j_k}$ correspond to the clauses using $\neg x_j$ (for $1 \leqslant j \leqslant m$).
- The twig query $q = r[//c_1] \ldots [//c_n]$.

For example, for the formula $\varphi_0 = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$ we obtain the DIMS S containing the rules:

$$r \to (t_1 \mid f_1) \parallel (t_2 \mid f_2) \parallel (t_3 \mid f_3) \parallel (t_4 \mid f_4),$$
$$t_1 \to c_1, \qquad f_1 \to c_2, \qquad t_2 \to \epsilon, \qquad f_2 \to c_1,$$
$$t_3 \to c_1 \parallel c_2, \qquad f_3 \to \epsilon, \qquad t_4 \to \epsilon, \qquad f_4 \to c_2.$$

and the query $q = /r[//c_1][//c_2]$. The formula φ is satisfiable iff $(S, q) \in \mathrm{SAT}_{DIMS,Twig}$. The described reduction works in polynomial time in the size of the input formula.

For the NP upper bound, we reduce $\mathrm{SAT}_{DIMS,Twig}$ to $\mathrm{SAT}_{DTD,Twig}$ (i.e., the problem of satisfiability of twig queries in the presence of DTDs), known to be in NP (Theorem 4.4 from [5]). Given a DIMS S, we construct a DTD D having the same root label as S and whose rules are obtained from the rules of S by replacing the unordered concatenation with standard (ordered) concatenation. Then, take a twig query q. We claim that there exists an (unordered) tree satisfying q and S iff there exists an (ordered) tree satisfying q and D. For the *if* part, take an ordered tree t satisfying q and D, remove the order to obtain an unordered tree t', and observe that t' satisfies S. For the *only if* part, take an unordered tree t satisfying q and S. From the construction of D, we infer that there exists $S' \in_{dims}$ an ordered tree t' (obtained via some ordering of the sibling nodes of t) satisfying both q and D. We recall that the twig queries disregard the relative order among the siblings. □

The complexity results for query implication and query containment in the presence of DIMSs follow from the EXPTIME-completeness proof from [35] for twig query containment in the presence of DTDs.

Proposition 4 $\mathrm{IMPL}_{DIMS,Twig}$ *and* $\mathrm{CNT}_{DIMS,Twig}$ *are EXPTIME-complete.*

Proof The EXPTIME-hardness proof of twig containment in the presence of DTDs (Theorem 4.5 from [35]) has been done using a reduction from the *Two-player corridor tiling* problem and a technique introduced in [32]. In the proof from [35], when testing the containment $p \subseteq_S q$, p is chosen such that it satisfies every tree in S, hence $\mathrm{IMPL}_{DTD,Twig}$ is also EXPTIME-complete. Furthermore, Lemma 3 in

[32] can be adapted to twig queries and DIMS: for every $S \in \mathit{DIMS}$ and twig queries $q_0, q_1, \ldots, q_m$ there exists $S' \in \mathit{DIMS}$ and twig queries q and q' such that $q_0 \subseteq_S q_1 \cup \ldots \cup q_m$ iff $q \subseteq_{S'} q'$. Moreover, the DTD in [35] can be captured with a DIMS constructible in polynomial time: take the same reduction as in [35] and then replace the standard concatenation with unordered concatenation. Hence, we infer that $\mathrm{CNT}_{\mathit{DIMS},\mathit{Twig}}$ and $\mathrm{IMPL}_{\mathit{DIMS},\mathit{Twig}}$ are also EXPTIME-hard.

For the EXPTIME upper bound, we reduce $\mathrm{CNT}_{\mathit{DIMS},\mathit{Twig}}$ to $\mathrm{CNT}_{\mathit{DTD},\mathit{Twig}}$ (i.e., the problem of twig query containment in the presence of DTDs), known to be in EXPTIME (Theorem 4.4 from [35]). Given a DIMS S, we construct a DTD D having the same root label as S and whose rules are obtained from the rules of S by replacing the unordered concatenation with standard (ordered) concatenation. Then, take two twig queries p and q. We claim that $p \subseteq_S q$ iff $p \subseteq_D q$ and show the two parts by contraposition. For the *if* part, assume $p \not\subseteq_S q$, hence there exists an unordered tree t that satisfies q and S, but not p. From the construction of D, we infer that there exists an ordered tree t' (obtained via some ordering of the sibling nodes of t) that satisfies q and D, but not p. For the *only if* part, assume $p \not\subseteq_D q$, hence there exists an ordered tree t that satisfies q and D, but not p. By removing the order of t, we obtain an unordered tree t' that satisfies q and S, but not p. We recall that the twig queries disregard the relative order among the siblings. The membership of $\mathrm{CNT}_{\mathit{DIMS},\mathit{Twig}}$ to EXPTIME yields that $\mathrm{IMPL}_{\mathit{DIMS},\mathit{Twig}}$ is also in EXPTIME (it suffices to take as p the universal query). □

7 Complexity of Disjunction-Free Interval Multiplicity Schemas (IMSs)

Although *query satisfiability* and *query implication* in the presence of schema are intractable for DIMSs, we prove that they become tractable for IMSs (Section 7.4). We also show a considerably lower complexity for *query containment* in the presence of schema: coNP-completeness for IMSs instead of EXPTIME-completeness for DIMSs (Section 7.4). Additionally, we point out that our results for IMSs allow also to characterize the complexity of query implication and query containment in the presence of *disjunction-free DTDs* (i.e., restricted DTDs using regular expressions without disjunction operator), which, to the best of our knowledge, have not been previously studied (Section 7.5). To prove our results, we develop a set of tools that we present next: *dependency graphs* (Section 7.1), *generalized definition of embedding* (Section 7.2), *family of characteristic graphs* (Section 7.3).

7.1 Dependency Graphs

Recall that IMSs use IMEs, which are essentially expressions of the form $A_1^{I_1} \parallel \ldots \parallel A_k^{I_k}$, where $A_1, \ldots, A_k$ are atoms, and $I_1, \ldots, I_k$ are intervals. Given an IME E, let $\mathit{symbols}^{\forall}(E)$ be the set of symbols present in all unordered words in $L(E)$, and $\mathit{symbols}(E)$ the set of symbols present in at least one unordered word in $L(E)$:

$$\mathit{symbols}^{\forall}(E) = \{a \in \Sigma \mid \forall w \in L(E).\ a \in w\},$$
$$\mathit{symbols}^{\exists}(E) = \{a \in \Sigma \mid \exists w \in L(E).\ a \in w\}.$$

Given an IME E, notice that $\mathit{symbols}^{\forall}(E) \subseteq \mathit{symbols}^{\exists}(E)$, and moreover, the sets $\mathit{symbols}^{\forall}(E)$ and $\mathit{symbols}(E)$ can be easily constructed from E. For example, given $E_0 = (a \| b^?)^{[5,6]} \| c^+$, we have $\mathit{symbols}^{\forall}(E_0) = \{a, c\}$ and $\mathit{symbols}(E_0) = \{a, b, c\}$.

Definition 2 Given an IMS $S = (\mathit{root}_S, R_S)$, the *existential dependency graph* of S is the directed rooted graph $G_S^{\exists} = (\Sigma, \mathit{root}_S, E_S^{\exists})$ with the node set Σ, the distinguished root node root_S, and the set of edges $E_S^{\exists}$ such that $(a, b) \in E_S^{\exists}$ if $b \in \mathit{symbols}^{\exists}(R_S(a))$. Furthermore, the *universal dependency graph* of S is the directed rooted graph $G_S^{\forall} = (\Sigma, \mathit{root}_S, E_S^{\forall})$ such that $(a, b) \in E_S^{\forall}$ if $b \in \mathit{symbols}^{\forall}(R_S(a))$.

Example 5 Take the IMS S containing the rules:

$$r \to (a^? \parallel b)^{[1,10]} \parallel c, \qquad a \to d^?, \qquad b \to a^{[2,3]} \parallel c^* \parallel d^+.$$

In Fig. 4 we present the existential dependency graph of S and the universal dependency graph of S. □

Given an IMS S and a symbol a, we say that a is *reachable* (or *useful*) in S if there exists a tree in $L(S)$ which has a node labeled by a. Moreover, we say that an IMS is *trimmed* if it contains rules only for the reachable symbols. For every satisfiable IMS S, there exists an equivalent trimmed version which can be obtained by removing the rules for the symbols involved in *unreachable components* in $G_S^{\forall}$ (in the spirit of [2]). Notice that the unreachable components of $G_S^{\forall}$ correspond in fact to cycles in $G_S^{\forall}$. In the sequel, we assume w.l.o.g. that all IMSs that we manipulate are satisfiable and trimmed.

7.2 Generalizing the Embedding

We generalize the notion of embedding previously defined in Section 2. Note that in the rest of the section we use the term *dependency graphs* when we refer to both existential and universal dependency graphs. First, an *embedding* of a query q in a dependency graph $G = (\Sigma, \mathit{root}, E)$ is a function $\lambda : N_q \to \Sigma$ such that:

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

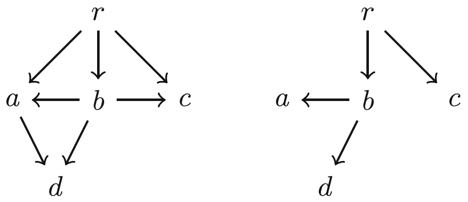


Fig. 4 Existential dependency graph $G_S^{\exists}$ and universal dependency graph $G_S^{\forall}$ for Example 5

If there exists an embedding of q in G, we write $G \preccurlyeq q$. Next, a *simulation* of a dependency graph $G = (\Sigma, root, E)$ in a tree t is a relation $R \subseteq \Sigma \times N_t$ such that:

1 $(root, root_t) \in R$,
2 for every $(a, n) \in R$, $(a, a') \in E$, there exists $n' \in N_t$ such that $(n, n') \in child_t$ and $(a', n') \in R$,
3 for every $(a, n) \in R$. $lab_t(n) = a$.

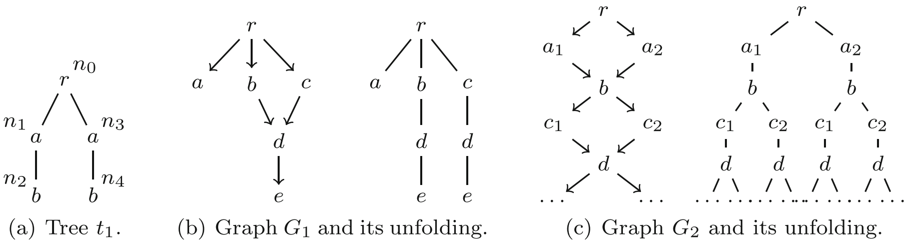
Note that R is a total relation for the nodes of the graph reachable from the root i.e., for every $a \in \Sigma$ reachable from $root$ in G, there exists a node $n \in N_t$ such that $(a, n) \in R$. If there exists a simulation from G to t, we write $t \preccurlyeq G$. Additionally, note that given a graph containing cycles reachable from the root, there does not exist any (finite) tree where it can be simulated. However, we point out that in the remainder we use the notion of simulation only for universal dependency graphs that are supposed to come from trimmed IMSs, hence they do not have such cycles.

Given two dependency graphs $G_1 = (\Sigma, root, E_1)$ and $G_2 = (\Sigma, root, E_2)$, G_1 is a *subgraph* of G_2 if $E_1 \subseteq E_2$. For a dependency graph $G = (\Sigma, root, E)$, we define the partial order $\leq_G$ on the subgraphs of G: given G_1 and G_2 two subgraphs of G, $G_1 \leq_G G_2$ if G_1 is a subgraph of G_2. Note that the relation $\leq_G$ is reflexive, antisymmetric, and transitive, thus being an ordering relation. Moreover, it is well-founded and it has a minimal element $G_0 = (\Sigma, root, \varnothing)$. The following result can be easily shown by a structural induction using the order $\leq_G$.

Lemma 6 *For every IMS S, its universal dependency graph can be simulated in every tree t which belongs to the language of S.*

A *path in a dependency graph* $G = (\Sigma, root, E)$ is a non-empty sequence of vertices starting at $root$ such that for every two consecutive vertices in the sequence, there is a directed edge between them in G. By $Paths(G) \subseteq \Sigma^+$ we denote the set of all paths in G. The set of paths is finite only for graphs without cycles reachable from the root. For instance, the paths of the graph G_1 in Fig. 5b are $Paths(G_1) = \{r, ra, rb, rc, rbd, rcd, rbde, rcde\}$.

Similarly, a *path in a tree t* is a non-empty sequence of nodes starting at $root_t$ such that every two consecutive nodes in the sequence are in the relation $child_t$. By $Paths(t) \subseteq N_t^+$ we denote the set of all paths in t. Then, we define $LabPaths(t) \subseteq \Sigma^+$ as the set of sequences of labels of nodes from all paths in t. For instance, for



(a) Tree t_1. (b) Graph G_1 and its unfolding. (c) Graph G_2 and its unfolding.

Fig. 5 A tree and two graphs with their corresponding unfoldings

the tree t_1 from Fig. 5b we have $Paths(t_1) = \{n_0, n_0n_1, n_0n_1n_2, n_0n_3, n_0n_3n_4\}$ and $LabPaths(t_1) = \{r, ra, rab\}$. Note that $|LabPaths(t)| \leq |Paths(t)|$. The *unfolding* of a dependency graph $G = (\Sigma, root, E)$, denoted u_G, is a tree $u_G = (N_{u_G}, root_{u_G}, lab_{u_G}, child_{u_G})$ such that:

- $N_{u_G} = Paths(G)$,
- $root_{u_G} \in N_{u_G}$ is the root of u_G,
- $(p, p \cdot a) \in child_{u_G}$, for every path $p, p \cdot a \in Paths(G)$ (note that "·" stands for standard ordered concatenation),
- $lab_{u_G}(root_{u_G}) = root$, and $lab_{u_G}(p \cdot a) = a$, for every path $p \cdot a \in Paths(G)$.

The unfolding of a graph is finite only when the graph has no cycle reachable from the root, because otherwise $Paths(G)$ is infinite, hence u_G is infinite. In the remainder, we use the unfolding only for graphs having no cycle reachable from the root (in order to have finite unfoldings). In such a case, the unfolding can be seen as the *smallest* tree u_G (w.r.t. the number of nodes) having $LabPaths(u_G) = Paths(G)$. The idea of the unfolding is to transform the dependency graph G into a tree having the *child* relation instead of directed edges. There are nodes duplicated in order to avoid nodes with more than one incoming edge. For instance, in Fig. 5b we take the graph G_1 and construct its unfolding u_{G_1}. Moreover, notice that the size of the unfolding may be exponential in the size of the graph, for example for the graph G_2 from Fig. 5c.

We also extend the definition of embedding and propose the embedding from a tree to another tree i.e., given two trees t and t', we say that t' can be embedded in t (denoted $t \preccurlyeq t'$) if the query $(N_{t'}, root_{t'}, lab_{t'}, child_{t'}, \emptyset)$ can be embedded in t. Similarly, we can define the embedding from a tree to a dependency graph. Note that two embeddings can be *composed*, for example:

- $\forall t, t' \in Tree.\ \forall q \in Twig.\ (t \preccurlyeq t' \wedge t' \preccurlyeq q \Rightarrow t \preccurlyeq q)$,
- $\forall S \in IMS.\ \forall t \in Tree.\ \forall q \in Twig.\ (G_S^{\forall/\exists} \preccurlyeq t \wedge t \preccurlyeq q \Rightarrow G_S^{\forall/\exists} \preccurlyeq q)$.

We state next two auxiliary lemmas that can be easily proven by structural induction on the dependency graphs (using the order $\leq_G$):

Lemma 7 *A dependency graph G can be simulated in a tree t iff its unfolding u_G can be embedded in t.*

Lemma 8 *A query q can be embedded in a dependency graph G iff q can be embedded in the unfolding tree of G.*

In Fig. 6 we present the operations *fuse* and *add*. Given two trees t and t', we say that $t \lhd_0 t'$ if t' is obtained from t by applying one of the operations from Fig. 6. The *fuse* operation takes two siblings with the same label and creates only one node having below it the subtrees corresponding to each of the siblings. The *add* operation consists simply in adding a subtree at some place in the tree. By $\unlhd$ we denote the transitive and reflexive closure of $\lhd_0$.

Note that the fuse and add operations preserve the embedding i.e., given a twig query q and two trees t and t', if $t \preccurlyeq q$ and $t \unlhd t'$, then $t' \preccurlyeq q$. Furthermore,

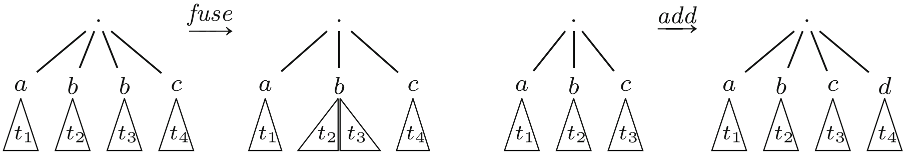


Fig. 6 Operations *fuse* and *add*

if we can embed a query q in a tree t which can be embedded in the existential dependency graph of an IMS S, we can perform a sequence of operations such that t is transformed into another tree t' satisfying S and q at the same time. Formally, we have the following.

Lemma 9 *Given an IMS S, a query q and a tree t, if $G_S^{\exists} \preccurlyeq t$ and $t \preccurlyeq q$, then there exists a tree $t' \in L(S) \cap L(q)$. The tree t' can be constructed after a sequence of fuse and add operations (consistently with the schema S) from the tree t and we denote $t \trianglelefteq_S t'$.*

7.3 Family of Characteristic Graphs

Given a schema S and a query q, we can capture all trees satisfying both S and q with the characteristic graphs that we introduce next.

More formally, a *characteristic graph G* is a tuple $(V_G, root_G, lab_G, E_G)$, where V_G is a finite set of vertices, $root_G \in V_G$ is the root of the graph, $lab_G : V_G \to \Sigma$ is a labeling function (with $lab_G(root_G) = root_S$), and $E_G \subseteq V_G \times V_G$ is the set of edges. Let us assume that $G_S^{\exists} \preccurlyeq q$ and take such an embedding $\lambda : N_q \to \Sigma$. By $\Lambda(q, S, \lambda)$ we denote the set of all *characteristic graphs for q and S w.r.t. λ*. To construct such a graph, let us start with $G = (V_G, root_G, lab_G, E_G)$ where V_G and E_G are empty, and perform the four steps described below.

1. For every n in N_q, add a node n' to V_G such that $lab_G(n') = \lambda(n)$. Let $root_G$ be the node such that $lab_G(root_G) = root_S$.
2. For every (n_1, n_2) in $child_q$, add (n_1', n_2') to E_G, where n_1' and n_2' are the nodes corresponding to n_1 and n_2, respectively, as constructed at step 1.
3. For every (n_1, n_2) in $desc_q$, *choose an acyclic path* $a_0, \ldots, a_k$ in $G_S^{\exists}$ where $\lambda(n_1) = a_0$ and $\lambda(n_2) = a_k$. Notice that, since n_1 and n_2 belong to N_q, we have already added in V_G two nodes n_1' and n_2', respectively, corresponding to them at step 1. Then, for every a_i (with $1 \leq i \leq k-1$), we add in V_G a node n_i'' such that $lab_G(n_i'') = a_i$. Also, add in E_G the edges (n_1', n_1''), (n_1'', n_2''), …, (n_{k-1}'', n_2').
4. For every n in V_G, take from $G_S^{\forall}$ the subgraph $(V', lab_G(n), E')$ rooted at $lab_G(n)$. Then, for every $a \neq lab_G(n)$ in V' add a node n' in V' such that $lab_G(n') = a$. Also, for every $(a_1, a_2) \in E'$, add in E_G an edge (n_1, n_2) where n_1 and n_2 are the nodes corresponding to a_1 and a_2, respectively.

The following example illustrates the construction of such a graph.

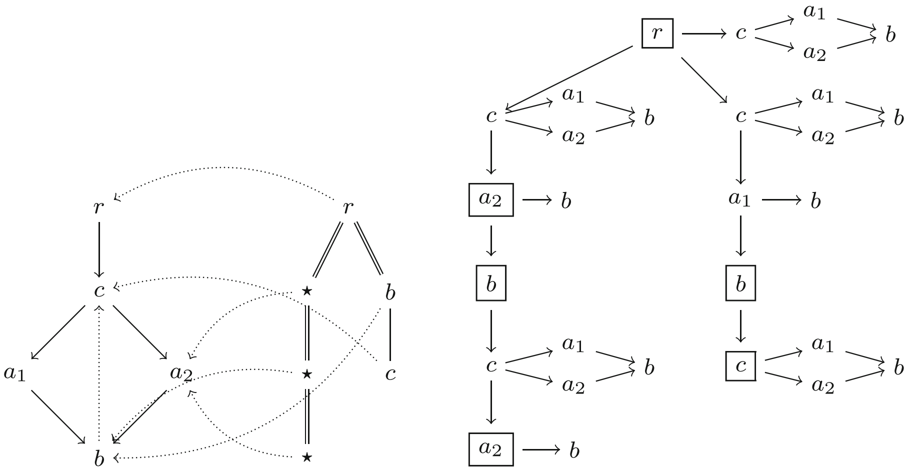
Example 6 Take in Fig. 7a an existential dependency graph $G^{\exists}_{S}$, a twig query q, and an embedding $\lambda : N_q \to G^{\exists}_{S}$. Notice that in $G^{\exists}_{S}$ we have drawn the universal edges with a full line and those that are existential without being universal with a dotted line. Then, in Fig. 7b we present an example of a graph G from $\Lambda(q, S, \lambda)$. Notice that in G we have represented in boxes the nodes corresponding to the images $\lambda(n)$ for the nodes of the query $n \in N_q$. □

Next, we define the set of all characteristic graphs for q and S w.r.t. the all embeddings λ of q in $G^{\exists}_{S}$:

$$\mathcal{G}(q, S) = \{G \in \Lambda(q, S, \lambda) \mid \lambda \text{ is an embedding of } q \text{ in } G^{\exists}_{S}\}.$$

Note that $G \preccurlyeq q$ and the size of G is polynomially bounded by $|q| \times |\Sigma|^2$ for every G in $\mathcal{G}(q, S)$. Indeed, after step 1 of the construction, a characteristic graph G has $|q|$ nodes. Then, after steps 2 and 3, since at step 3 we allow only acyclic paths of $G^{\exists}_{S}$, we add at most $|\Sigma|$ nodes for each already existing node, hence G has at most $|q| \times |\Sigma|$ nodes. Finally, after 4, since we add at most $|\Sigma|$ nodes for each already existing node, G has at most $|q| \times |\Sigma|^2$ nodes.

Furthermore, let $\Lambda^*(q, S, \lambda)$ and $\mathcal{G}^*(q, S)$ be sets of characteristic graphs constructed similarly to $\Lambda(q, S, \lambda)$ and $\mathcal{G}(q, S)$, respectively, the only difference being that we allow cyclic paths at step 3 of the aforementioned construction. While the size of the graphs in $\mathcal{G}(q, S)$ is polynomial, notice that the size of the graphs in $\mathcal{G}^*(q, S)$ is not necessary polynomial since the possible cyclic paths chosen at step 3 can be arbitrarily long. Additionally, note that $|\mathcal{G}(q, S)|$ is finite and may be exponential while $|\mathcal{G}^*(q, S)|$ may be infinite if the existential dependency graph $G^{\exists}_{S}$ contains cycles reachable from the root.



(a) Embedding $\lambda : N_q \to G^{\exists}_{S}$.

(b) Graph $G \in \Lambda(q, S, \lambda)$.

Fig. 7 An embedding from a query q to an existential dependency graph $G^{\exists}_{S}$ and a graph $G \in \mathcal{G}(q, S)$. In $G^{\exists}_{S}$, the universal edges are drawn with a full line and those that are existential without being universal with a dotted line

Next, we extend the previous definition of the unfolding to the characteristic graphs. Given an IME E and a symbol a, by $min\_nb(E, a)$ we denote the minimum number of occurrences of the symbol a in every unordered word defined by E. Next, we define the *unfolding of a characteristic graph*. Given a query q, an IMS S, and a characteristic graph $G \in \mathcal{G}^*(q, S)$, we construct its unfolding as follows:

- Let u_G be the unfolding of G obtained as defined in Section 7.2.
- Update u_G such that for every $n \in N_{u_G}$, for every $a \in \Sigma$, let t_a the subtree having as root the child of n labeled by a. Next, add copies of t_a as children of n until n has $min\_nb(R_S(lab_{u_G}(n)), a)$ children labeled by a.

Notice that every graph G in $\mathcal{G}^*(q, S)$ is acyclic. Indeed, when constructing such a graph G, after steps 1, 2 and 3, G is basically shaped as a tree. Then, the subgraphs that we fuse at step 4 are all acyclic since they are subgraphs of the universal dependency graph $G_S^\forall$ that we assume trimmed (cf. Section 7.1). Since every graph G in $\mathcal{G}^*(q, S)$ is acyclic, it has a finite unfolding, which naturally belongs to the language of S.

7.4 Complexity Results

In this section, we use the above defined tools to show the complexity results for IMSs. First, the dependency graphs and embeddings capture satisfiability and implication of queries by IMSs.

Lemma 10 *Given a twig query q and an IMS S:*

1. *q is satisfiable by S iff $G_S^\exists \preccurlyeq q$,*
2. *q is implied by S iff $G_S^\forall \preccurlyeq q$.*

Proof *1)* For the *if* part, we know that $G_S^\exists \preccurlyeq q$, thus the family of graphs $\mathcal{G}(q, S)$ is not empty. The unfolding of every graph from $\mathcal{G}(q, S)$ satisfies S and q at the same time, hence q is satisfiable by S. For the *only if* part, we know that there exists a tree $t \in L(S) \cap L(q)$, and we assume w.l.o.g. that it is the unfolding of a graph G from $\mathcal{G}^*(q, S)$. Since $t \preccurlyeq q$, we obtain $u_G \preccurlyeq q$, hence $G \preccurlyeq q$ (by Lemma 8), which, from the construction of G, implies that $G_S^\exists \preccurlyeq q$.

2) For the *if* part, we know that $G_S^\forall \preccurlyeq q$, which implies by Lemma 8 that $u_{G_S^\forall} \preccurlyeq q$. On the other hand, take a tree $t \in L(S)$. By Lemma 6 we have $t \preccurlyeq G_S^\forall$, which implies by Lemma 7 that $t \preccurlyeq u_{G_S^\forall}$. From the last embedding and $u_{G_S^\forall} \preccurlyeq q$ we infer that $t \preccurlyeq q$. Since t can be every tree in the language of S, we conclude that q is implied by S. For the *only if* part, we know that for every $t \in L(S)$, $t \preccurlyeq q$. Consider the tree t obtained as follows: we take $u_{G_S^\forall}$ and we duplicate some subtrees in order to have, for each node $n \in N_t$, $min\_nb(R_S(lab_t(n)), a)$ children labeled by a. Naturally, t is in the language of S, hence $t \preccurlyeq q$ from the hypothesis. From the definition of the unfolding, we infer that $G_S^\forall \preccurlyeq t$, which implies that $G_S^\forall \preccurlyeq q$. □

For instance, the twig query $q = r[a]/b//d$ can be embedded in the existential dependency graph of the IMS S from Example 5, thus q is satisfiable by S. In Fig. 8

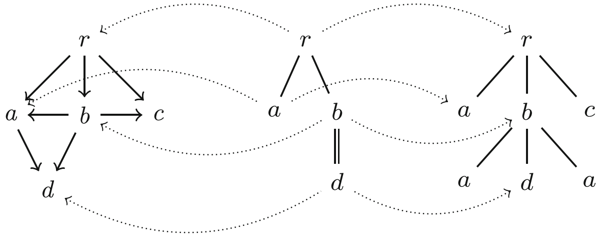


Fig. 8 Embeddings of q in $G_S^{\exists}$ and in a tree t which satisfies S and q at the same time

we present embeddings of q in $G_S^{\exists}$ and in a tree t satisfying both S and q. Additionally, notice that the twig query $q = r[a]/b//d$ cannot be embedded in $G_S^{\forall}$ from Example 5, and therefore, q is not implied by S. On the other hand, the twig query $q' = r/b//d$ can be embedded in $G_S^{\forall}$, thus q' is implied by S.

Moreover, we point out that testing the embedding of a query in a dependency graph can be done in polynomial time with a simple bottom-up algorithm. From this observation and Lemma 10 we obtain the following.

Theorem 5 $\mathrm{SAT}_{\mathit{IMS,Twig}}$ *and* $\mathrm{IMPL}_{\mathit{IMS,Twig}}$ *are in PTIME.*

Next, we present the complexity of query containment in the presence of IMSs. The coNP-completeness of the containment of twig queries [32] implies the coNP-hardness of the containment of twig queries in the presence of IMSs. Proving the membership of the problem to coNP is, however, not trivial. Given an instance (p, q, S), the set of all trees satisfying p and S can be characterized with a set $\mathcal{G}(p, S)$ 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 graphs in $\mathcal{G}(p, S)$. This condition is easily checked by a non-deterministic Turing machine.

Theorem 6 $\mathrm{CNT}_{\mathit{IMS,Twig}}$ *is coNP-complete.*

Proof The coNP-completeness of the containment of twig queries (Theorem 4 in [32]) implies that $\mathrm{CNT}_{\mathit{IMS,Twig}}$ is coNP-hard. Next, we prove the membership of the problem to coNP. Given an instance (p, q, S), a witness is a function $\lambda : N_p \to \Sigma$. Testing whether λ is an embedding from p to $G_S^{\exists}$ requires polynomial time. If λ is an embedding, a non-deterministic polynomial algorithm chooses a graph G from $\Lambda(p, S, \lambda)$ and checks whether q can be embedded in G. We claim that $p \not\subseteq_S q$ iff there exists a graph G in $\mathcal{G}(p, S)$ such that $G \not\preccurlyeq q$.

For the *if* case, we assume that there exists a graph $G \in \mathcal{G}(p, S)$ such that $G \not\preccurlyeq q$. We know that $G \preccurlyeq p$, thus $u_G \preccurlyeq p$ (by Lemma 8), hence there exists a tree $t \in L(S)$ such that $t \preccurlyeq p$ and $u_G \trianglelefteq_S t$ (by Lemma 9). If we assume by absurd that $t \preccurlyeq q$, we have $u_G \preccurlyeq q$, thus $G \preccurlyeq q$, which is a contradiction. We infer thus that there exists a tree $t \in L(S) \cap L(p)$, such that $t \notin L(q)$, and consequently, $p \not\subseteq_S q$.

For the *only if* case, we assume that $p \not\subseteq_S q$, hence there exists a tree $t \in L(S) \cap L(p)$ such that $t \notin L(q)$. Because $t \in L(S) \cap L(p)$, we know that there exists a graph

$G \in \mathcal{G}^*(p, S)$, such that $u_G \trianglelefteq_S t$. We know that $t \not\preccurlyeq q$, thus $u_G \not\preccurlyeq q$ (by Lemma 8), that yields $G \preccurlyeq q$. Furthermore, by using a simple pumping argument, we have $\forall q \in \mathit{Twig}.\ \forall G \in \mathcal{G}^*(q, S).\ (G \not\preccurlyeq q \Rightarrow \exists G' \in \mathcal{G}(q, S).\ G' \not\preccurlyeq q)$, which implies that there exists a graph $G' \in \mathcal{G}(p, S)$ such that $G' \not\preccurlyeq q$. □

7.5 Extending the Complexity Results to Disjunction-Free DTDs

We also point out that the complexity results for implication and containment of twig queries in the presence of IMSs can be adapted to disjunction-free DTDs. This allows us to state results which, to the best of our knowledge, are novel. Similarly to the IMSs, we represent a *disjunction-free DTD* as a tuple $S = (root_S, R_S)$, where $root_S$ is a designated root label and R_S maps symbols to regular expressions using no disjunction, basically regular expressions of the grammar:

$$E ::= \epsilon \mid a \mid E^* \mid E^? \mid E^+ \mid (E \cdot E),$$

where $a \in \Sigma$ and "·" stands for the standard concatenation operator. Given such an expression E, let $\mathit{symbols}^\forall(E)$ be the set of symbols present in all words from $L(E)$, and $\mathit{symbols}(E)$ the set of symbols present in at least one word from $L(E)$:

$$\mathit{symbols}^\forall(E) = \{a \in \Sigma \mid \forall w \in L(E).\ \exists w_1, w_2.\ w = w_1 \cdot a \cdot w_2\},$$
$$\mathit{symbols}^\exists(E) = \{a \in \Sigma \mid \exists w \in L(E).\ \exists w_1, w_2.\ w = w_1 \cdot a \cdot w_2\}.$$

As pointed out for the IMEs, note that the sets $\mathit{symbols}^\forall(E)$ and $\mathit{symbols}^\exists(E)$ can be easily constructed from E. Next, we adapt the notions of dependency graph and universal dependency graph for disjunction-free DTDs. The *existential dependency graph* of a disjunction-free DTD S is a directed rooted graph $G_S^\exists = (\Sigma, root_S, E_S^\exists)$, where

$$E_S^\exists = \{(a, a') \mid a' \in \mathit{symbols}^\exists(R_S(a))\}.$$

Similarly, the *universal dependency graph* of a disjunction-free DTD S is a directed rooted graph $G_S^\forall = (\Sigma, root_S, E_S^\forall)$, where

$$E_S^\forall = \{(a, a') \mid a' \in \mathit{symbols}^\forall(R_S(a))\}.$$

Analogously to the IMSs, we assume w.l.o.g. that we manipulate only disjunction-free DTDs having no cycle reachable from the root in the universal dependency graph. Otherwise, if there is a cycle in the universal dependency graph, this means that there is no tree consistent with the schema and containing at least one of the symbols implied in that cycle. Moreover, similarly to IMSs, for a symbol $a \in \Sigma$ and a disjunction-free regular expression E, by $min\_nb(E, a)$ we denote the minimum number of occurrences of the symbol a in every word defined by E.

Next, we state our complexity results for disjunction-free DTDs.

Theorem 7 $\mathrm{IMPL}_{\textit{disj-free-DTD,Twig}}$ *is in PTIME and* $\mathrm{CNT}_{\textit{disj-free-DTD,Twig}}$ *is coNP-complete.*

Proof We claim that a query q is implied by a disjunction-free DTD S iff $G_S^\forall \preccurlyeq q$ and since the embedding of a query in a graph can be computed in polynomial time, this implies that $\mathrm{IMPL}_{\textit{disj-free-DTD},\textit{Twig}}$ is in PTIME. The proof follows from the proof of Lemma 10.2. The coNP-completeness of the containment of twig queries (Theorem 4 in [32]) implies that $\mathrm{CNT}_{\textit{disj-free-DTD},\textit{Twig}}$ is coNP-hard. Theorem 6 states the coNP-completeness of the query containment in the presence of IMSs and an easy adaptation of its proof technique yields the membership of $\mathrm{CNT}_{\textit{disj-free-DTD},\textit{Twig}}$ to coNP. The mentioned proofs can be adapted because given a disjunction-free regular expression E and a word $u \in L(E)$, u can in fact be obtained as an ordering of the unordered word $w = \biguplus_{a\in\Sigma} a^{min\_nb(E,a)}$. Moreover, the order imposed by the DTD on the siblings is not important because the twig queries are order-oblivious. □

8 Expressiveness of DIMS

First, we compare the expressive power of DIMSs with yardstick languages of unordered trees. We begin with FO logic that uses only the binary *child* predicate and the unary label predicates P_a with $a \in \Sigma$. It is easy to show that DIMSs are not comparable with FO. With a simple rule $a \to (b \parallel c)^*$ a DIMS can express the language of trees where every node labeled by a has as children only nodes labeled by b and c such that the number of b's is equal to the number of c's. Such language cannot be captured with FO for reasons similar to those for which it cannot be expressed in FO whether the cardinality of the universe is even. There are languages of unordered trees expressible by FO, but not expressible by DIMSs e.g., the language of trees that contain exactly two nodes labeled b. Such languages are not expressible by DIMSs for reasons similar to those for which they cannot be expressed by DTDs, more precisely they are not closed under substitution of subtrees with the same root type (cf. Lemma 2.10 in [37]). By using exactly the same examples, note that DIMSs and MSO are also incomparable. MSO with Presburger constraints [12, 13, 43, 44] is essentially an extension of MSO that additionally allows elements of arithmetic (numerical variables and value comparisons) and unary functions $\#a$ that return the number of children of a node having a given label $a \in \Sigma$. This extension is very powerful and can express Parikh images of arbitrary regular languages. DIMSs are strictly less expressive than Presburger MSO as they use a strict restriction of unordered regular expressions.

Next, we compare the expressive power of DIMSs and DTDs. For this purpose, we introduce a simple tool for comparing regular expressions with DIMEs. Given 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 DIME E *captures* R if $L(E) = L(R)$. This tool is equivalent to considering DTDs under *commutative closure* [4, 34]. 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 DIME that captures a regular expression defines basically the same admissible content model of a node, without imposing an order among the children.

Naturally, by using the above notion to compare the expressive powers of DTDs and DIMSs, DTDs are strictly more expressive than DIMSs. For example, the commutative closure of the regular expression $(a \cdot (b \mid c))^*$ cannot be expressed by a DIME. Various classes of regular expressions have been reported in widespread use in real-world schemas and have been studied in the literature: *simple regular expressions* [8, 29], *single occurrence regular expressions* (SOREs) [7], *chain regular expressions* (CHAREs) [7]. DIMEs are strictly more expressive than CHAREs and incomparable to the other mentioned classes of regular expressions.

Finally, we investigate how many real-life DTDs can be captured with DIMSs and use the comparison on the XMark benchmark [39] and the University of Amsterdam XML Web Collection [23]. All 77 regular expressions of the XMark benchmark are captured by DIMEs, and among them 76 by IMEs. As for the DTDs from the University of Amsterdam XML Web Collection, 92 % of regular expressions are captured by DIMEs and among them 78 % by IMEs. We also point out that CHAREs, captured by DIMEs, are reported to represent up to 90 % of regular expressions used in real-life DTDs [7]. These numbers give a generally positive coverage, but should be interpreted with caution, as we do not know which of the considered DTDs were indeed intended for data-centric applications.

9 Related Work

Languages of unordered trees can be expressed by *logic formalisms* or by *tree automata*. Boneva et al. [12, 13] 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* [43], *sheaves automata* [20], and the *TQL logic* [15] allow to express *Presburger constraints* on the numbers of occurrences of the different symbols among the children of some node. Suitable restrictions allow to obtain the same expressiveness as the *Presburger MSO logic* on unordered trees [12, 13], strictly more expressive than DIMSs. Additionally, we believe that DIMSs 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.

Languages of unordered trees can be also expressed by considering DTDs under *commutative closure* [4, 34]. We assume DTDs using arbitrary regular expressions, not necessarily one-unambiguous [14] as required by the W3C. We also point out that it has been recently shown that it is PSPACE-complete to decide whether a given regular expression can be rewritten as an equivalent one-unambiguous one [19]. Given a DTD using arbitrary regular expressions under commutative closure, we say that an (ordered) tree matches such a DTD iff every tree obtained by reordering of sibling nodes also matches the DTD. However, it is PSPACE-complete to test whether a DTD defines a commutatively-closed set of trees [34] and, moreover, such a DTD may be of exponential size w.r.t. the size of the alphabet, which makes such DTDs unfeasible. Another consequence of the high expressive power of DTDs under commutative closure is that the membership problem is NP-complete [26]. Therefore, these formalisms were not extensively used in practice. From a different point of view, Martens et al. [27, 28] investigate DTDs equipped with formulas from the $\mathcal{SL}$

logic that specifies unordered languages and obtain complexity improvements for typechecking XML transformations.

The unordered concatenation operator "$\parallel$" should not be confused with the *shuffle* (*interleaving*) operator "&" used in a restricted form in XML Schema and RELAX NG to define order-oblivious, yet still ordered, content. On the one hand, $a^*\&b$ defines all ordered words with an arbitrary number of a's and exactly one occurrence of b, and analogously, $a^* \parallel b$ defines all unordered words with exactly the same characteristic. On the other hand, $(a\&b)^*$ defines ordered words of the form $w_1 \cdot \ldots \cdot w_n$, where the factors $w_1, \ldots, w_n$ are either ab or ba, while $(a \parallel b)^*$ defines unordered words having the same number of a's and b's. For instance, $(a\&b)^*$ does not accept the ordered word $aabb$ while it has the same number of a's and b's. Adding the shuffle and interval multiplicities to the regular expressions increases the computational complexity of fundamental decision problems such as: membership [6, 25], inclusion, equivalence, and intersection [21]. Colazzo et al. [17, 18, 22] propose efficient algorithms for membership and inclusion of *conflict-free types*, a class of regular expressions with shuffle and numerical constraints using intervals. Their approach is based on capturing a language with a set of constraints, similar to our characterizing tuples for DIMEs. While conflict-free types and DIMEs both forbid repetitions of symbols, they differ on the restrictions imposed on the use of the operators and the interval multiplicities. Consequently, they are incomparable.

We finally point out that the static analysis problems involving twig queries i.e., twig query satisfiability [5], implication [9, 24], and containment [35] in the presence of schema have been extensively studied in the context of DTDs. However, to the best of our knowledge, these problems have not been previously studied neither for the mentioned unordered schema languages, nor for DTDs using classes of regular expressions extended with counting and interleaving.

10 Conclusions and Future Work

We have studied schema languages for unordered XML. First, we have investigated languages of unordered words and we have proposed disjunctive interval multiplicity expressions (DIMEs), a subclass of unordered regular expressions for which two fundamental decision problems, membership of an unordered word to the language of a DIME and containment of two DIMEs, are tractable. Next, we have employed DIMEs to define languages of unordered trees and have proposed disjunctive interval multiplicity schema (DIMS) and its restriction, disjunction-free interval multiplicity schema (IMS). DIMSs and IMSs can be seen as DTDs using restricted classes of regular expressions and interpreted under commutative closure to define unordered content models. These restrictions allow to maintain a relatively low computational complexity of basic static analysis problems while allowing to capture a significant part of the expressive power of practical DTDs.

As future work, we want to study whether the restrictions imposed by the grammar of DIMEs can be relaxed while maintaining the tractability of the problems of interest. Moreover, we would like to investigate learning algorithms for the unordered schema languages proposed in this paper. We have already proposed learning algorithms for restrictions of DIMSs and IMSs [16] and we want to extend them to take into account all the expressive power. We also aim to apply the unordered schemas to query minimization [3] i.e., given a query and a schema, find a smaller yet equivalent query in the presence of the schema. Furthermore, we want to use unordered schemas and optimization techniques to boost the learning algorithms for twig queries [45].

References

1. Abiteboul, S., Bourhis, P., Vianu, V.: Highly expressive query languages for unordered data trees. In: ICDT, pp. 46–60 (2012)
2. Albert, J., Giammarresi, D., Wood, D.: Normal form algorithms for extended context-free grammars. Theor. Comput. Sci. **267**(1-2), 35–47 (2001)
3. Amer-Yahia, S., Cho, S., Lakshmanan, L.V.S., Srivastava, D.: Tree pattern query minimization. VLDB J. **11**(4), 315–331 (2002)
4. Beeri, C., Milo, T.: Schemas for integration and translation of structured and semi-structured data. In: ICDT, pp. 296–313 (1999)
5. Benedikt, M., Fan, W., Geerts, F.: XPath satisfiability in the presence of DTDs. J. ACM **55** (2) (2008)
6. Berglund, M., Björklund, H., Högberg, J.: Recognizing shuffled languages. In: LATA, pp. 142–154 (2011)
7. Bex, G.J., Neven, F., Schwentick, T., Vansummeren, S.: Inference of concise regular expressions and DTDs. ACM Trans. Database Syst **35**(2) (2010)
8. Bex, G.J., Neven, F., Van den Bussche, J.: DTDs versus XML Schema A practical study. In: WebDB, pp. 79–84 (2004)
9. Björklund, H., Martens, W., Schwentick, T.: Validity of tree pattern queries with respect to schema information MFCS, pp. 171–182 (2013)
10. Boneva, I., Ciucanu, R., Staworko, S.: Simple schemas for unordered XML. In: WebDB, pp. 13–18 (2013)
11. Boneva, I., Gayo, J.E.L., Hym, S., Prud'hommeau, E.G., Solbrig, H.R., Staworko, S.: Validating RDF with shape expressions. arXiv:1404.1270 CoRR (2014)
12. Boneva, I., Talbot, J.: Automata and logics for unranked and unordered trees. In: RTA, pp. 500–515 (2005)
13. Boneva, I., Talbot, J., Tison, S.: Expressiveness of a spatial logic for trees. In: LICS, pp. 280–289 (2005)
14. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Inf. Comput. **142**(2), 182–206 (1998)
15. Cardelli, L., Ghelli, G.: TQL: a query language for semistructured data based on the ambient logic. Math. Struct. Comput. Sci. **14**(3), 285–327 (2004)
16. Ciucanu, R., Staworko, S.: Learning schemas for unordered XML. In: DBPL, pp. 31–40 (2013)
17. Colazzo, D., Ghelli, G., Pardini, L., Sartiani, C.: Almost-linear inclusion for XML regular expression types. ACM Trans. Database Syst. **38**(3), 15 (2013)
18. Colazzo, D., Ghelli, G., Sartiani, C.: Efficient inclusion for a class of XML types with interleaving and counting. Inf. Syst. **34**(7), 643–656 (2009)
19. Czerwinski, W., David, C., Losemann, K., Martens, W.: Deciding definability by deterministic regular expressions. In: FoSSaCS, pp. 289–304 (2013)
20. Dal-Zilio, S., Lugiez, D.: XML schema, tree logic and sheaves automata RTA, pp. 246–263 (2003)
21. Gelade, W., Martens, W., F. Neven.: Optimizing schema languages for XML, Numerical constraints and interleaving. SIAM J. Comput. **38**(5), 2021–2043 (2009)
22. Ghelli, G., Colazzo, D., Sartiani, C.: Linear time membership in a class of regular expressions with interleaving and counting. In: CIKM, pp. 389–398 (2008)
23. Grijzenhout, S., Marx, M.: The quality of the XML web. J. Web Sem. **19**, 59–68 (2013)
24. Hashimoto, K., Kusunoki, Y., Ishihara, Y., Fujiwara, T.: Validity of positive XPath queries with wildcard in the presence of DTDs. In: DBPL (2011)

25. Hovland, D.: The membership problem for regular expressions with unordered concatenation and numerical constraints. In: LATA, pp. 313–324 (2012)
26. Kopczynski, E., To, A.: Parikh images of grammars Complexity and applications. In: LICS, pp. 80–89 (2010)
27. Martens, W., Neven, F.: On the complexity of typechecking top-down XML transformations. Theor. Comput. Sci. **336**(1), 153–180 (2005)
28. Martens, W., Neven, F., Gyssens, M.: Typechecking top-down XML transformations Fixed input or output schemas. Inf. Comput. **206**(7), 806–827 (2008)
29. Martens, W., Neven, F., Schwentick, T.: Complexity of decision problems for simple regular expressions. In: MFCS, pp. 889–900 (2004)
30. Martens, W., Neven, F., Schwentick, T.: Complexity of decision problems for XML schemas and chain regular expressions. SIAM J. Comput. **39**(4), 1486–1530 (2009)
31. Mayer, A.J., Stockmeyer, L.J.: Word problems-this time with interleaving. Inf. Comput. **115**(2), 293–311 (1994)
32. Miklau, G., Suciu, D.: Containment and equivalence for a fragment of XPath. J. ACM **51**(1), 2–45 (2004)
33. Montazerian, M., Wood, P.T., Mousavi, S.R.: XPath query satisfiability is in PTIME for real-world DTDs XSym, pp. 17–30 (2007)
34. Neven, F., Schwentick, T.: XML schemas without order (1999)
35. Neven, F., Schwentick, T.: On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. Logical Methods in Computer Science **2**(3) (2006)
36. Oppen, D.C.: A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. J. Comput. Syst. Sci. **16**(3), 323–332 (1978)
37. Papakonstantinou, Y., Vianu, V.: DTD inference for views of XML data. In: PODS, pp. 35–46 (2000)
38. Schaefer, T.J.: The complexity of satisfiability problems. In: STOC, pp. 216–226 (1978)
39. Schmidt, A., Waas, F., Kersten, M., Carey, M., Manolescu, I., XMark, R.Busse.: A benchmark for XML data management VLDB, pp. 974–985 (2002)
40. Schwentick, T.: Trees, automata and XML PODS, p. 222 (2004)
41. Segoufin, L., Sirangelo, C.: Constant-memory validation of streaming XML documents against DTDs. In: ICDT, pp. 299–313 (2007)
42. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: PODS, pp. 53–64 (2002)
43. Seidl, H., Schwentick, T., Muscholl, A.: Numerical document queries. In: PODS, pp. 155–166 (2003)
44. Seidl, H., Schwentick, T., Muscholl, A.: Counting in trees. In: Logic and Automata, pp. 575–612 (2008)
45. Staworko, S., Wieczorek, P.: Learning twig and path queries. In: ICDT, pp. 140–154 (2012)
46. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time Preliminary report STOC, pp. 1–9 (1973)
47. W3C: XML Path language (XPath) 1.0 (1999)