Schemas for Unordered XML on a DIME

Iovka Boneva  Radu Ciucanu  Slawek Staworko
University of Lille & INRIA, France
{iovka.boneva, radu.ciucanu, slawomir.staworko}@inria.fr

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 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.

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

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 author’s. A book may additionally contain one publisher and may also have one or more editor’s instead of author’s. A schema has numerous important uses. For instance, it allows to validate a document against a schema and identify potential errors. A schema also serves as a reference for any user who does not know yet the structure of the XML document and attempts to query or modify its contents.

The Document Type Definition (DTD), the most widespread XML schema formalism for (ordered) XML [7, 20], 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. For instance, the DTD below corresponds to the one used in practice for the DBLP repository:

dbpl → (article | book)*
article → (title | year | author)*
book → (title | year | author | editor | publisher)*
This DTD allows an article to contain any number of title, year, and author elements. A book may also have any number of title, year, author, editor, and publisher elements. These regular expressions are clearly over-permissive because they allow XML documents that do not follow the intuitive guidelines set out earlier e.g., an XML document containing an article with two title’s and no author 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 [2, 40]:

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

The first query selects the elements labeled book, children of dblp and having an author containing the text “C. Papadimitriou.” The second query additionally requires that book has a title. Naturally, these two queries should be equivalent because every book element should have a title child. 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. We consider unordered regular expressions, which are essentially regular expressions with unordered concatenation “[ ]” instead of standard concatenation. 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 its unordered content model. For instance, take the following schema (satisfied by the tree in Figure 1):

```
dblp ∈ article* || book*
article → title || year || author+
book → title || year || publisher|| (author+ | editor+)
```

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 editor’s instead of author’s. Note that, unlike the DTD defined earlier, this schema does not allow documents having an article with several title’s or without any author.

Using UREs is equivalent to using DTDs with regular expressions interpreted under the commutative closure [3, 30]: 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 [22] 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 [39, 27]. 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|a” because the symbol a is used twice. Instead we add general interval multiplicities a[l: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 remains to be characterized, 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 (DISM), and its restriction, disjunction-free interval multiplicity schema (IMS). Naturally, the above schema for the DBLP repository is a DISM. 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 and we observe that DISMs 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 a normal form for DIEMEs in the form of characterizing tuples that can be computed in polynomial time and allow deciding in polynomial time the membership of a tree to the language of a DIMS and 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.

### Organization

In Section 2 we introduce some preliminary notions while in Section 3 we present languages of unordered words. In Section 4 we define the problems of interest and we analyze their complexity for the proposed schema languages. In Section 5 we discuss the expressiveness of the proposed formalisms. In Section 6 we summarize our results and outline further directions. Because of space restrictions, we omit several proofs, which can be found in the appendix. This work is an extended version of a preliminary work presented in [9].

### Related word

Languages of unordered trees can be expressed by logic formalisms or by tree automata. Boneva et al. [10, 11] 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 [36], sheaves automata [17], and the TQL logic [13] allow to express Presburger constraints on the numbers of occurrences of the different symbols among the children of some node. This is also equivalent to considering DTDs under commutative closure, similarly to [3, 30]. The consequence of the high expressive power is that the membership problem is NP-complete [22]. Therefore, these formalisms were not extensively used in practice. Suitable restrictions on Presburger automata and on the TQL logic allow to obtain the same expressiveness as the MSO logic on unordered trees [10, 11]. DIMSs are strictly less expressive than these MSO-equivalent languages. 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. As mentioned earlier, unordered content model can also be defined by DTDs defining commutatively-closed sets of ordered trees. An (ordered) tree matches such a DTD iff all tree obtained by reordering of sibling nodes also matches the DTD. This also turns out to be equally expressive as MSO on unordered trees [10, 11]. However, such a DTD may be of exponential size w.r.t. the size of the alphabet and, moreover, it is PSPACEm-complete to test whether a DTD defines a commutatively-closed set of trees [30], which makes such DTDs unfeasible. From a different point of view, Martens et al. [23, 24] investigate DTDs which allow on the rhs on the rules formulas from the SL logic that specifies unordered languages and obtain complexity improvements for typechecking XML transformations.

The unordered concatenation operator “||” 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 model. On the one hand, $a^\ast \& b$ defines all ordered words with an arbitrary number of $a$’s and exactly one occurrence of $b$, and analogously, $a^\ast \| b$ defines all unordered words with exactly the same characteristic. On the other hand, $(a\&b)^\ast$ defines ordered words of the form $w_1 \cdots w_n$, where the factors $w_1, \ldots, w_n$ are either $ab$ or $ba$, while $(a \| b)^\ast$ defines unordered words having the same number of $a$’s and $b$’s. For instance, $(a\&b)^\ast$ does not accept the ordered word $aabb$ while it has the same number of $a$’s and $b$’s. Adding the shuffle and in-
tential multiplicities to the regular expressions increases
the computational complexity of fundamental decision
problems such as: membership [5, 21], inclusion, equi-
valence, and intersection [18]. Ghelli et al. [15, 16, 19]
propose efficient algorithms for membership and inclu-
sion 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 normal form for DI-
MEs. 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.

To the best of our knowledge, the static analysis pro-
blems involving twig queries have not been previously
studied neither for the MSO-equivalent languages, nor
for DTDs using classes of regular expressions extended
with counting and interleaving.

2. PRELIMINARIES

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

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

\[
\begin{align*}
\text{Tree} & = \{ (N, \text{root}, \text{lab}, \text{child}) \mid \\
& \quad N \text{ is a finite set of nodes, root } \in N, \\
& \quad \text{lab} : N \to \Sigma, \\
& \quad \text{child} \subseteq N \times N \}
\end{align*}
\]

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

Figure 2: A tree and a twig query.

Queries. We work with the class of twig queries, which
are essentially unordered trees whose nodes may be ad-
ditionally labeled with a distinguished wildcard symbol
\( * \notin \Sigma \) and that use two types of edges, child (\( /\) ) and
descendant (\( //\) ), corresponding to the standard XPath
axes. Note that the semantics of //--edge is that of a
proper descendant (and not that of descendant-or-self).

Formally, a twig query \( q \) is a tuple \( (N_q, \text{root}_q, \text{lab}_q, \text{child}_q, \text{desc}_q) \), where \( N_q \) is a finite set of nodes, \( \text{root}_q \in N_q \) is the root node, \( \text{lab}_q : N_q \to \Sigma \cup \{\ast\} \) is a labeling
function, \( \text{child}_q \subseteq N_q \times N_q \) is a set of child edges, and
\( \text{desc}_q \subseteq N_q \times N_q \) is a set of descendant edges. We
assume that \( \text{child}_q \cup \text{desc}_q = \emptyset \) and that the relation
\( \text{child}_q \cup \text{desc}_q \) is acyclic and we require every non-root
node to have exactly one predecessor in this relation.

By Twig we denote the set of all twig queries. Twig qu-
eries are often presented using the abbreviated XPath
syntax [40] e.g., the query \( q_0 \) in Figure 2(b) can be written
as \( r//\{\ast\}/a \).

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

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

If there exists an embedding of \( q \) in \( t \) we say that \( t \)
satisfies \( q \) 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 Figure 2.

\[
\begin{align*}
\text{Figure 3: Embeddings of } q_0 \text{ in } t_0.
\end{align*}
\]

3. UNORDERED REGULAR EXPRESSIONS

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 \( \varepsilon \) is an unordered word that has 0 occur-
cences 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 ti-
mes 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 \( w_0(c) = 2 \). Addi-
tionally, we may write \( w_0 = a^3c^2 \) instead of \( w_0 = aaacc \).

We use unordered words to model collections of chil-
dren of XML nodes. It is natural to assume that every
node takes the same amount of memory, and thus, we
use a unary representation of unordered words, where each occurrence of a symbol occupies the same amount of space. 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 \textit{(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 all \( a \in \Sigma \). For instance, \( aaacc \uplus abbc = aaaaabbcucc \). Note that \( \varepsilon \) is the identity element of the unordered concatenation \( \varepsilon \uplus w = w \uplus \varepsilon = w \) for all unordered word \( w \).

Also, given an unordered word \( w \), by \( w^i \) we denote the concatenation \( w \ldots w \) (\( i \) times).

A \textit{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 \( L_1 = \{ a, aa \} \) 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 \textit{unordered regular expression} (URE) defines unordered words by using Kleene star \( "^*" \), disjunction \( "\|" \), and unordered concatenation \( \"^\|\" \).

Formally,

\[
E ::= \varepsilon | a | E^* | (E^\| E) | (E^\| E),
\]

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

\[
L(\varepsilon) = \{ \varepsilon \},
\]

\[
L(a) = \{ a \},
\]

\[
L(E_1 \| E_2) = L(E_1) \cup L(E_2),
\]

\[
L(E_1 ^\| E_2) = L(E_1) \uplus L(E_2),
\]

\[
L(E^*) = \{ w_1 \| \ldots \| w_{i} \mid w_1, \ldots, w_i \in L(E) \land i \geq 0 \}.
\]

For instance, the URE \( (a \| (b \| c))^* \) 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 \textit{multiplicity} \( ^* \) and we introduce macros for two other standard and commonly used multiplicities:

\[
E^+ ::= E \| E^*, \quad E^? ::= E \| \varepsilon.
\]

The URE \( (a \| b^* \| (a \| c))^? \) accepts the unordered words having at least one \( a \), more \( a \)'s than \( b \)'s, possibly one \( c \).

Next, we study two fundamental decision problems for UREs: membership of an unordered word to the language of a URE and containment of two UREs. As we show, using arbitrary UREs easily yields the intractability of the two problems. Consequently, we shall propose \textit{disjunctive interval multiplicity expressions} (DIMEs), a restriction which enjoys better computational properties.

3.1 Intractability results

It can be easily seen that deciding the membership of an unordered word to the language of a URE can be reduced to testing the membership of a vector to the Parikh image of a regular language, known to be NP-complete [22], and vice versa. We show, however, 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 [22].

\textbf{Theorem 3.1} Given an unordered word \( w \) and an expression \( E \) of the grammar \( E ::= a \| E^? | (E^\| E) \), deciding whether \( w \in L(E) \) is NP-complete.

\textbf{Proof.} We show only the NP-hardness and prove it by reduction from SAT_{1-in-3}, known as being NP-complete [33]. We take a 3CNF formula \( \varphi = c_1 \land \ldots \land c_k \) over the variables \( \{ x_1, \ldots, x_n \} \). We take the alphabet \( \{ d_1, \ldots, d_k, v_1, \ldots, v_n \} \) and we construct the unordered word \( w_\varphi = d_1 \ldots d_k v_1 \ldots v_n \). Next, we construct the expression \( E_\varphi = X_1 \| \ldots \| X_n \), where for \( 1 \leq j \leq n \):

\[
X_j = (v_j \| d_{1j} \| \ldots \| d_{kj})^? \| (v_j \| d_{1j} \| \ldots \| d_{km})^?.
\]

and \( d_{1j}, \ldots, d_{kj} \) correspond to the clauses which use the literal \( x_j \), and \( d_{1j}, \ldots, d_{km} \) correspond to the clauses which use the literal \( \neg x_j \). For example, for the formula \( \varphi_0 = (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_3 \lor \neg x_4) \), we construct \( w_{\varphi_0} = d_1 d_2 v_1 v_3 v_4 \) and

\[
E_{\varphi_0} = (v_1 \| d_1)^? \| (v_1 \| d_2)^? \| (v_2 \| d_1)^? \| (v_3 \| d_1 \| d_2)^? \| (v_3 \| d_2)^? \| (v_4 \| d_2)^?.
\]

We claim that \( \varphi \in \text{SAT}_{1\text{-in-3}} \) iff \( w_{\varphi} \in L(E_{\varphi}) \). For the only if case, we use the \( \text{SAT}_{1\text{-in-3}} \) valuation of \( \varphi \) to construct the derivation of \( w_{\varphi} \in L(E_{\varphi}) \). For the if case, we assume that \( w_{\varphi} \in L(E_{\varphi}) \). Since \( w_{\varphi}(v_j) = 1 \), we infer that \( w_{\varphi} \) uses exactly one of the expressions of the form \( (v_j \| \ldots)^? \). Moreover, since \( w_{\varphi}(d_i) = 1 \), we infer that the valuation encoded in \( w_{\varphi} \) validates exactly one literal of each clause in \( \varphi \), and therefore, \( \varphi \in \text{SAT}_{1\text{-in-3}} \). Clearly, the described reduction works in polynomial time.

While regular expression containment is a PSPACE-complete problem [39], the order plays an essential role in the reduction, which cannot be adapted to characterize the complexity of the containment of UREs. We prove independently that deciding the containment of UREs is \( \Pi_2^P \)-hard and we show an upper bound which follows from the complexity of deciding the satisfiability of Presburger logic formulas [6, 37].

\textbf{Theorem 3.2} Given two UREs \( E_1 \) and \( E_2 \), deciding \( L(E_1) \subseteq L(E_2) \) is 1 \( \Pi_2^P \)-hard and 2 \( \text{in-EXPTIME} \).
Proof. 1) We prove the \(\Pi^p_2\)-hardness by reduction from the problem of checking the satisfiability of \(\forall^p\exists^p\text{-}\text{QBF}\) formulas, a classical \(\Pi^p_2\)-complete problem. We take a \(\forall^p\exists^p\text{-}\text{QBF}\) formula

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

where \(\varphi = c_1 \land \ldots \land c_k\) 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, \(E_{\psi}\) and \(E_{\psi}'\). First, \(E_{\psi} = d_1 \mid \ldots \mid d_k \mid X_1 \mid \ldots \mid X_n\), where for \(1 \leq i \leq n\)

\[
X_i = ((t_i \mid d_i) \mid \ldots \mid (t_i \mid d_i)) \mid ((f_i \mid d_i) \mid \ldots \mid (f_i \mid d_i)),
\]

and \(d_i, \ldots, d_i\) correspond to the clauses which use the literal \(x_i\), and \(d_i, \ldots, d_i\) correspond to the clauses which use the literal \(\neg x_i\).

For example, for the formula

\[
\psi_0 = \forall x_1, x_2. \exists y_1, y_2. \quad (x_1 \lor \neg x_2 \lor y_1) \land (\neg x_1 \lor y_1 \lor \neg y_2) \land (x_2 \lor \neg y_1),
\]

we construct:

\[
E_{\psi_0} = d_1 \mid d_2 \mid d_3 \mid (t_1 \mid d_1) \mid \ldots \mid (t_1 \mid d_1) \mid (f_1 \mid d_1) \mid \ldots \mid (f_1 \mid d_1).
\]

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 \(\psi_0\), the unordered word \(d_1^2 d_3 d_2 t_1 f_2\) corresponds to the valuation \(V\) s.t. \(V(x_1) = \text{true}\) and \(V(x_2) = \text{false}\).

Next, we construct \(E_{\psi}' = X_1 \mid \ldots \mid X_n \mid Y_1 \mid \ldots \mid Y_m\), where:

- \(X_i = ((t_i \mid d_i) \mid \ldots \mid (t_i \mid d_i)) \mid (f_i \mid d_i) \mid \ldots \mid (f_i \mid d_i)\), and \(d_i, \ldots, d_i\) correspond to the clauses which use the literal \(x_i\) (for \(1 \leq i \leq n\)),

- \(Y_j = ((d_j^1 \mid \ldots \mid d_j^m) \mid (d_j^1 \mid \ldots \mid d_j^m))\), and \(d_j, \ldots, d_j\) correspond to the clauses which use the literal \(y_j\) (for \(1 \leq j \leq m\)).

For example, for \(\psi_0\) above we construct:

\[
E_{\psi_0}' = ((t_1 \mid d_1) \mid (f_1 \mid d_2)) \mid ((t_2 \mid d_3) \mid (f_2 \mid d_1)) \mid ((d_1^1 \mid d_1^2) \mid (d_1^1 \mid d_1^2)).
\]

We claim that \(\models \psi\) iff \(E_{\psi} \leq E_{\psi}'\). 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 \(\varphi\), we use this valuation to construct a derivation of \(w\) in \(L(E_{\psi}')\). For the if case, for any 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 \(\varphi\). 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 [32].

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 [32]. Given two UREs \(E_1\) and \(E_2\), we compute in linear time [37] two existential Presburger formulas for their Parikh images: \(\varphi_{E_1}\) and \(\varphi_{E_2}\), respectively. Next, we test the satisfiability of the following closed Presburger logic formula:

\[
\exists \overline{x}. \varphi_{E_1}(\overline{x}) \Rightarrow \varphi_{E_2}(\overline{x}).
\]

The proofs of Theorem 3.1 and Theorem 3.2 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 (SO-REs) [8], conflict-free types [15, 16, 19], and duplicate-free DTDs [29].

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 \(E[1,n]\) and \(E[1,m]\), where \(n \in \mathbb{N}_0\) and \(m \in \mathbb{N}_0\). Their semantics is defined as follows:

\[
L(E[1,n], n \leq i \leq m) = \{w_i \mid w_i \in L(E) \land n \leq i \leq m\},
\]

\[
L(E[1,m]) = L(E[1,m]) \cup \{\varepsilon\}.
\]

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:

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

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

While the complexity of the membership of an unordered word to the language of a URE with intervals and without symbol repetitions remains to be characterized, testing containment of two such UREs continues to be intractable.

Theorem 3.3 Given two UREs \(E_1\) and \(E_2\) with intervals and 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 SAT. Take a CNF formula \(\varphi = c_1 \land \ldots \land c_k\) over the variables \(\{x_1, \ldots, x_n\}\). Take the alphabet \(\{a_{ij} | 1 \leq i \leq k, 1 \leq j \leq n, c_i\text{ uses } x_j \text{ or } \neg x_j\}\). We construct the expression \(E_{\varphi} = X_1 \mid \ldots \mid X_n\), where \(X_j = ((a_{i1} \mid \ldots \mid a_{ij}) \mid (a_{f1} \mid \ldots \mid a_{fj})\) (for \(1 \leq j \leq n\)), and \(c_1, \ldots, c_i\) are the clauses which use the literal \(x_j\), and \(c_{f1}, \ldots, c_{fj}\) are the clauses which use the literal \(\neg x_j\). Next, we construct \(E_{\varphi} = \left(C_1 \mid \ldots \mid C_k\right)[0, k-1]\), where \(C_i = (a_{ij1} \mid \ldots \mid a_{ijp}^+)\) (for \(1 \leq i \leq k\), and
$x_1, \ldots, x_n$ are the variables used by the clause $c_i$. For example, for 
\[ \varphi_0 = (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (x_2 \lor \neg x_3), \]
we obtain:
\[ E_{x_1} = (\varphi_0 \mid a_{12}) \mid (a_{12} \mid a_{13}) \mid (a_{13} \mid a_{23}) \mid (a_{23} \mid a_{24}) \mid (a_{24} \mid a_{04}) \]
\[ E_{x_2} = (a_{11} \mid a_{12} \mid a_{13} \mid a_{23} \mid a_{24} \mid (a_{12} \mid a_{23}) \mid (a_{23} \mid a_{04}) \mid (0.2). \]
Note that there is an one-to-one correspondence between the unordered words in $L(E_{x_i})$ and the valuations of the variables $x_1, \ldots, x_n$ (*). For example, for above $\varphi_0$ and the valuation $V$ s.t. $V(x_1) = V(x_2) = V(x_3) = true$ and $V(x_4) = false$, the unordered word $w_V = a_{11} a_{12} a_{13} a_{23} a_{24}$ belongs to $L(E_{x_0})$. Moreover, given an $w_V \in L(E_{x_i})$, 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 (**): $w \in L(E_{x'})$ iff there is $i \in \{1, \ldots, k\}$ s.t. none of the symbols used in $C_i$ occurs in $w$.

From (*) and (**), we infer that given a valuation $V$, $V \models \varphi$ iff $w_V \in L(E_{x_i}), L(E_{x'})$, that yields $\varphi \in SAT$ iff $L(E_{x_i}) \subseteq L(E_{x'})$. Clearly, the described reduction works in polynomial time.

Theorem 3.3 shows that disallowing repetitions of symbols in a URE with intervals does not avoid the intractability of the containment. In the next section, we impose further restrictions, that yield a class of UREs with desirable computational properties.

### 3.2 Disjunctive interval multiplicity expressions (DIMEs)

An atom is $(a_1^1 || \ldots || a_k^i)$, where all $I_i$’s are ? or 1. For example, $(a \mid b \mid c)$ is an atom, but $(a \mid a \mid a)$ is not an atom. A clause is $(A_1^1 || \ldots || A_k^i)$, 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, $(a \mid (b \mid c))$ is a clause (which is not simple), $(a \mid (b \mid c))$ is a simple clause while $(a \mid (b \mid c))$ is not a clause.

A disjunctive interval multiplicity expression (DIME) is $(D_1^1 || \ldots || D_k^i)$, where for $1 \leq i \leq k$ either 1) $D_i$ is a simple clause and $I_i \in \{+, \ast\}$, 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, $(a \mid (b \mid c))$ is a DIME while $(a \mid b \mid c)$ is not a DIME because it uses the symbol a twice. A disjunctive-free interval multiplicity expression (IME) is a DIME which does not use the disjunction operator. An example of IME is $a \mid (b \mid c)$ for more practical examples of DIMEs see Examples 4.2 and 4.3.

Next, we introduce an alternative representation of DIMEs. Recall that $a \in w$ is short for $w(a) = 0$. Given a DIME $E$, we define the characterizing tuple $\Delta_E = (C_E, N_E, P_E, K_E)$ consisting of the following sets:

- The conflicting pairs of siblings $C_E$ consisting of pairs of symbols in $\Sigma$ s.t. $E$ defines no word using both symbols simultaneously:
  \[ C_E = \{(a, b) \in \Sigma \times \Sigma \mid \exists w \in L(E). a \in w \land b \notin w\}. \]
- The extended cardinality map $N_E$ captures for each symbol in the alphabet the possible numbers of its occurrences in the unordered words defined by $E$:
  \[ N_E = \{(a, w(a)) \in \Sigma \times \mathbb{N}_0 \mid w \in L(E)\}. \]
- The collections of required symbols $P_E$ which captures symbols that must be present in every word; essentially, a set of symbols $X$ belongs to $P_E$ if every word defined by $E$ contains at least one element from $X$:
  \[ P_E = \{X \subseteq \Sigma \mid \forall w \in L(E). w(a) \geq w(b)\}. \]
- The counting dependencies $K_E$ consisting of pairs of symbols $(a, b)$ s.t. there is no unordered word defined by $E$ having more $b$’s than $a$’s. 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.
  \[ K_E = \{(a, b) \in \Sigma \times \Sigma \mid \forall w \in L(E). w(a) = w(b)\}. \]

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

\[ C_{E_0} = \{(b, d), (c, d), (d, b), (d, c)\}, \]
\[ N_{E_0} = \{(b, i) \mid i \geq 1\} \cup \{(b, i) \mid i \geq 0\} \cup \{(c, i) \mid i \geq 0\} \cup \{(d, i) \mid i = 0 \lor i \geq 5\}, \]
\[ P_{E_0} = \{\{a\}, \{b, d\}\ldots\}, \]
\[ K_{E_0} = \{(b, c)\}. \]

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 reasoning. First, however, we present the properties of the characterizing tuples that allow us to use them to reason about the languages defined by the corresponding DIMEs.

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

1. $\forall (a, b) \in C_E. (a \in w \Rightarrow b \notin w) \land (b \in w \Rightarrow a \notin w)$,
2. $\forall a \in \Sigma. (a, w(a)) \in N_E$, 
3. $\forall X \in P_E. \exists a \in X. a \in w$,
4. $\forall (a, b) \in K_E. w(a) \geq w(b)$.

We show that the characterizing tuple $\Delta_E$ captures precisely the language of $E$. 

\[ L(E) = \{w(a) \geq w(b) \mid (a, b) \in C_E\}. \]
Lemma 3.4 Given an unordered word \( w \) and a DIME \( E \), \( w \in L(E) \) iff \( w \models \Delta_E \).

Essentially, the language defined by an expression comprises of all unordered words satisfying the characterizing tuple. We illustrate the above lemma on an example. For instance, the unordered word \( aabbcde \) belongs to the language of the DIME \( E_0 = a^* \| (b \| c^+ \| d^{[5,\infty]} ) \) since it satisfies all the four conditions imposed by the characterizing tuple. On the other hand, note that the following unordered words do not belong to the language of \( E_0 \):

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

Next, we define the subsumption of two characterizing tuples. Given two DIMEs \( E \) and \( E' \), we write \( \Delta_{E'} \subseteq \Delta_E \) if \( C_E \subseteq C_{E'} \), \( N_{E'} \subseteq N_E \), \( P_{E'} \subseteq P_E \), and \( K_{E'} \subseteq K_E \). Moreover, the subsumption of the characterizing tuples allows us to capture the containment of DIMEs.

Lemma 3.5 Given two DIMEs \( E \) and \( E' \), \( L(E) \subseteq L(E') \) iff \( \Delta_{E'} \subseteq \Delta_E \).

Example 3.6 For the following DIMEs, it always holds that \( L(E') \subseteq L(E) \) and \( L(E) \nsubseteq L(E') \):

- Take \( E = a^* \| b^+ \) and \( E' = (a \| b^+)^* \). Note that \( K_E = \emptyset \) and \( K_{E'} = \{(a,b)\} \). For instance, the unordered word \( b \) belongs to \( L(E) \), but does not belong to \( L(E') \);
- Take \( E = a^{[3,6]} \| b^+ \) and \( E' = a^{[3,6]} \| b^+ \). Note that \( P_E = \emptyset \), and \( P_{E'} = \{(a,b)\} \). For instance, the unordered word \( e \) belongs to \( L(E) \), but does not belong to \( L(E') \);
- Take \( E = (a \| b^+)^* \) and \( E' = (a \| b^+)^{[0,5]} \). Note that \( (a,6) \in N_E \), but not \( (a,5) \). For instance, the unordered word \( ab \) belongs to \( L(E) \), but does not belong to \( L(E') \);
- Take \( E = (a \| b^+) \) and \( E' = a^+ \| b^+ \). Note that \( C_E = \emptyset \), 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 3.5 shows that two equivalent DIMEs yield the same characterizing tuple and hence the tuple \( \Delta_E \) can be viewed as a normal form for the language definable by a DIME \( E \). Formally,

Corollary 3.7 Given two DIMEs \( E \) and \( E' \), \( L(E) = L(E') \) iff \( \Delta_E = \Delta_{E'} \).

We now focus on compact representation of characterizing tuples. While \( N_E \) may be infinite, it can be easily represented in a compact manner using intervals: for any 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 \( N_E^R(a) \) we denote the interval representing the set \( \{ i \in \mathbb{N}_0 \mid (a,i) \in N_E \} \). For instance, for the DIME \( E_0 \) above, we obtain the function \( N_{E_0}^R \) below instead of \( N_{E_0} \):

\[
N_{E_0}^R(a) = [5, \infty],
N_{E_0}^R(b) = *,
N_{E_0}^R(c) = *,
N_{E_0}^R(d) = [5, \infty].
\]

Naturally, testing the inclusion \( N_{E'} \subseteq N_E \) can be easily reduced to a simple test on \( N_{E'}^R \) and \( N_E^R \).

Representing \( P_E \) in a compact manner is more tricky. A natural idea would be to store only its \( \ll \)-minimal elements. For instance, for the DIME \( E_1 = (a\|b) \| (c\|d) \) the set \( P_{E_1} \) has \( 4 \) \( \ll \)-minimal elements \( \{a,c\}, \{b,c\}, \{a,d\}, \) and \( \{b,d\} \), and the example easily generalizes to arbitrary numbers of atoms used in the clauses. We take a different approach, where we view an element \( x \in P_E \) as disjunctive formula \( \bigvee_{a \in x} a \in w \) e.g., \( \{a,c\} \) represents the formula \( a \in w \lor c \in w \). Because DIMEs allow the nesting of unordered concatenation in clauses (i.e., disjunctions of atoms), to represent in a compact manner the set \( P_E \) we shall use disjunctions of conjunctions of basic atoms of the form \( a \in w \). Such formulas are obtained from a DIME in a straightforward fashion: essentially, in every clause of the DIME we replace \( \| \) by \( \land \) and \( \| \) by \( \lor \) except for fragments that use intervals allowing 0 occurrences (a detailed description of this procedure can be found in Appendix).

Theorem 3.8 Given an unordered word \( w \) and two DIMEs \( E \) and \( E' \):

1. deciding \( w \in L(E) \) is in PTIME;
2. deciding \( L(E') \subseteq L(E) \) is in PTIME.
4. INTERVAL MULTIPLICITY SCHEMAS

First, we define the proposed schema languages.

Definition 4.1 A disjunctive interval multiplicity schema (DIMS) is a tuple \( S = (\text{root}_S, R_S) \), where \( \text{root}_S \in \Sigma \) is a designated root label and \( R_S \) maps symbols in \( \Sigma \) to DIMEs. By DIMS we denote the set of all disjunctive interval multiplicity schemas. A disjunction-free interval multiplicity schema (IMS) \( S = (\text{root}_S, R_S) \) is a restricted DIMS, where \( R_S \) maps symbols in \( \Sigma \) 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 \( ch^n_t \) of children of a node \( n \in N_t \) of \( t \), i.e., \( ch^n_t(a) = \{ \{ m \in N_t \mid (n, m) \in \text{child}_t \land \text{lab}_t(m) = a \} \} \). Now, a tree \( t \) satisfies \( S \), in symbols \( t \models S \), if \( \text{lab}_t(\text{root}_t) = \text{root}_S \) and for any node \( n \in N_t \), \( ch^n_t \in L(R_S(\text{lab}_t(n))) \). By \( L(S) \subseteq \text{Tree} \) we denote the set of all trees satisfying \( S \).

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

Example 4.2 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 he uploads, and 2) peers are split into two groups: a peer is a vip if he uploads at least 100 files, otherwise it is a simple user:

\[
\begin{align*}
\text{peers} & \rightarrow \text{user}^* \parallel \text{vip}^*, \\
\text{user} & \rightarrow (\text{upload} \parallel \text{download})^{[0,99]}, \\
\text{vip} & \rightarrow (\text{upload} \parallel \text{download})^{[100,\infty]}.
\end{align*}
\]

Example 4.3 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.

\[
\begin{align*}
\text{events} & \rightarrow \text{event}^*, \\
\text{event} & \rightarrow \text{date} \parallel ((\text{play} \parallel \text{theater}) \mid (\text{movie} \parallel \text{cinema})).
\end{align*}
\]

4.1 Problems of interest

We define the problems of interest and we formally state the corresponding decision problems parameterized by the class of schema \( S \) and, when appropriate, by a class of queries \( Q \).

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

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

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

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

Schema containment – checking if every given schema satisfies another given schema:

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

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

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

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

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

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

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

We next study these problems for DIMSs and IMs.

4.2 Disjunctive interval multiplicity schemas (DIMSs)

In this section we present the complexity results for DIMSs. A simple algorithm based on dynamic programming can decide the satisfiability of a DIMS. For deciding the membership of a tree \( t \) to the language of a DIMS \( S \), we can easily design a streaming algorithm with earliest rejection. The algorithm works for any arbitrary ordering of sibling nodes and uses space in \( O(\text{height}(t) \times |\Sigma|^2) \), where \( \text{height}(t) \) is the height of \( t \) defined in the usual way. Moreover, testing the containment of two DIMSs reduces to testing, for each symbol in the alphabet, the containment of the associated DTDs, which is in PTIME (Theorem 3.8). From all these observations, we obtain

Proposition 4.4 \( \text{SAT}_{\text{DIMS}}, \text{MEMB}_{\text{DIMS}}, \text{and} \text{CNT}_{\text{DIMS}} \) are in PTIME.

We continue with complexity results that follow from known facts. Query satisfiability for DTDs is known to be NP-complete [4] and we adapt the result for DIMS.

Proposition 4.5 \( \text{SAT}_{\text{DIMS},\text{Twig}} \) is NP-complete.

The complexity results for query implication and query containment in the presence of DIMSs follow from the EXPTIME-completeness proof from [31] for twig query containment in the presence of DTDs.

Proposition 4.6 \( \text{IMPL}_{\text{DIMS},\text{Twig}} \) and \( \text{CNT}_{\text{DIMS},\text{Twig}} \) are EXPTIME-complete.
4.3 Disjunction-free interval multiplicity schemas (IMSs)

In this section we present the static analysis for IMSs. Although query satisfiability and query implication are intractable for DIMSs, these problems become tractable for IMSs because they can be reduced to testing embedding of queries in some dependency graphs that we define below. Recall that IMSs essentially use expressions of the form $A_1^{i_1} \cdots A_k^{i_k}$, where $A_1, \ldots, A_k$ are atoms, and $I_1, \ldots, I_k$ are intervals.

Given an IME $E$, let $\text{non-nullable}(E)$ be the set of symbols present in all unordered words from $L(E)$, and $\text{nullable}(E)$ the set of symbols present in at least one unordered word from $L(E)$:

\[
\text{non-nullable}(E) = \{a \in \Sigma \mid \forall w \in L(E), a \in w\}, \\
\text{nullable}(E) = \{a \in \Sigma \mid \exists w \in L(E), a \in w\}.
\]

Note that the sets $\text{non-nullable}(E)$ and $\text{nullable}(E)$ can be easily constructed from $E$. For example, given $E_0 = (a \parallel b^{1,6} \parallel c^+)$, we have $\text{non-nullable}(E_0) = \{a, c\}$ and $\text{nullable}(E_0) = \{a, b, c\}$.

**Definition 4.7** Given an IMS $S = (\text{root}_S, R_S)$, the dependency graph of $S$ is the directed rooted graph $G_S = (\Sigma, \text{root}_S, E_S)$ with the node set $\Sigma$, the distinguished root node $\text{root}_S$, and the set of edges $E_S$ s.t. $(a, b) \in E_S$ if $b \in \text{nullable}(R_S(a))$. Furthermore, the universal dependency graph of $S$ is the directed rooted graph $G^\Sigma_S = (\Sigma, \text{root}_S, E^\Sigma_S)$ s.t. $(a, b) \in E^\Sigma_S$ if $b \in \text{non-nullable}(R_S(a))$.

**Example 4.8** Take the IMS $S$ containing the rules:

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

In Figure 4 we present the dependency graph of $S$ and the universal dependency graph of $S$.

---

**Figure 4:** Dependency graph $G_S$ and universal dependency graph $G^\Sigma_S$ of the IMS $S$.

Given an IMS $S$ and a symbol $a$, we say that $a$ is productive in $S$ if there exists a tree in $L(S)$ which has a node labeled by $a$. Moreover, we say that an IMS is pruned if it contains rules only for the productive symbols. For any satisfiable IMS $S$, there exists an equivalent pruned version which can be obtained by removing the rules for the symbols involved in cyclic components in $G^\Sigma_S$. In the sequel, we assume w.l.o.g. that all IMSs that we manipulate are satisfiable and pruned.

Next, we generalize the notion of embedding previously defined in Section 2. An embedding of a query $q$ in a directed rooted graph $G = (\Sigma, \text{root}, E)$ (which can be either a dependency graph or a universal dependency graph) is a function $\lambda : N_q \rightarrow \Sigma$ s.t.:

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

If there exists an embedding of $q$ in $G$, we write $G \preceq q$. The dependency graphs and embeddings capture satisfiability and implication of queries by IMSs.

**Lemma 4.9** Given a twig query $q$ and an IMS $S$:

1. $q$ is satisfiable by $S$ iff $G_S \preceq q$,
2. $q$ is implied by $S$ iff $G^\Sigma_S \preceq q$.

For instance, the twig query $q = r[a] \parallel b/d$ can be embedded in the dependency graph of the IMS $S$ from Example 4.8, and therefore $q$ is satisfiable by $S$. In Figure 5 we present embeddings of $q$ in $G_S$ and in a tree $t$ which satisfies $S$ and $q$ at the same time.

---

**Figure 5:** Embeddings of $q$ in $G_S$ and in a tree $t$ which satisfies $S$ and $q$ at the same time.

Note that the twig query $q = r[a] \parallel b/d$ cannot be embedded in $G^\Sigma_S$, and therefore, $q$ is not implied by $S$. On the other hand, the twig query $q' = r/b \parallel d$ can be embedded in $G^\Sigma_S$, and in fact, $q'$ is implied by $S$.

Furthermore, testing the embedding of a query in a graph can be done in polynomial time with a simple bottom-up algorithm. Using Lemma 4.9 we obtain

**Theorem 4.10** $\text{SAT}_{\text{IMS, Twig}}$ and $\text{IMPL}_{\text{IMS, Twig}}$ are in $\text{PTIME}$.

The coNP-completeness of the containment of twig queries [28] 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 $G(p, S)$.
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 \( G(p, S) \). This condition is easily checked by a non-deterministic Turing machine.

**Theorem 4.11** CNT\(_{\text{IMS,Twig}}\) is \( \text{coNP-complete} \).

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.

**Theorem 4.12** IMPL\(_{\text{disj-free/DTD,Twig}}\) is in \( \text{PTIME} \) and CNT\(_{\text{disj-free/DTD,Twig}}\) is \( \text{coNP-complete} \).

### 5. EXPRESSIVENESS OF DIMS

First, we compare the expressive power of the DIMSs with yardstick languages of unordered trees. We begin with FO logic that uses only the binary \( \text{child} \) 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 \( a \rightarrow (b \parallel c)^{\ast} \) a DIMS can express the language of trees where any node labeled by \( a \) has as children only nodes labeled by \( b \) and \( c \) s.t. 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 testing whether the carnality of the universe is even cannot be expressed in FO. There are languages of unordered trees expressible by FO, but not expressible by DIMSs e.g., any node labeled by \( a \) has exactly two children, both labeled by \( b \) s.t. each of them has exactly one child; for one of them, the child is labeled by \( c \) and for the other one the child is labeled by \( d \).

Here is essentially an extension of FO that additionally allows elements of arithmetic (numerical variables and value comparisons) and the 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 in fact, is strictly more expressive than DIMSs. Not surprisingly, MSO on unordered trees [10, 11] is also more expressive than DIMSs.

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 [3, 30].

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 any 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. 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 [7, 25], single occurrence regular expressions (SOREs) [8], chain regular expressions (CHAREs) [8]. 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 [34] and the University of Amsterdam XML Web Collection [20]. 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 [8]. 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.

### 6. 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 investigate learning algorithms for the unordered schema languages proposed in this paper. We have already proposed learning algorithms for restrictions of DIMSs and IMSs [14] 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 [2] 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 [38].

7. REFERENCES

[1] S. Abiteboul, P. Bourhis, and V. Vianu. Highly expressive query languages for unordered data trees. In ICDT, pages 46–60, 2012.
[2] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. VLDB J., 11(4):315–331, 2002.
[3] C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In ICDT, pages 296–313, 1999.
[4] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. J. ACM, 55(2), 2008.
[5] M. Berglund, H. Björklund, and J. Högborg. Recognizing shuffled languages. In LATA, pages 142–154, 2011.
[6] L. Berman. The complexity of logical theories. Theor. Comput. Sci., 11:71–77, 1980.
[7] G. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: A practical study. In WebDB, pages 79–84, 2004.
[8] G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. ACM Trans. Database Syst., 35(2), 2010.
[9] I. Boneva, R. Ciucanu, and S. Staworko. Simple schemas for unordered XML. In WebDB, 2013.
[10] I. Boneva and J. Talbot. Automata and logics for unranked and unordered trees. In RTA, pages 500–515, 2005.
[11] I. Boneva, J. Talbot, and S. Tison. Expressiveness of a spatial logic for trees. In LICS, pages 280–289, 2005.
[12] A. Br üggemann-Klein and D. Wood. One-ambiguous regular languages. Inf. Comput., 142(2):182–206, 1998.
[13] L. Cardelli and G. Ghelli. TQL: a query language for semi-structured data based on the ambient logic. Mathematical Structures in Computer Science, 14(3):285–327, 2004.
[14] R. Ciucanu and S. Staworko. Learning schemas for unordered XML. In DBPL, 2013.
[15] D. Colazzo, G. Ghelli, L. Pardini, and C. Sartiani. Linear inclusion for XML regular expression types. In CIKM, pages 137–146, 2009.
[16] D. Colazzo, G. Ghelli, and C. Sartiani. Efficient inclusion for a class of XML types with interleaving and counting. Inf. Syst., 34(7):643–656, 2009.
[17] S. Dal-Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In RTA, pages 246–263, 2003.
[18] W. Gelade, W. Martens, and F. Neven. Optimizing schema languages for XML: Numerical constraints and interleaving. SIAM J. Comput., 38(5):2021–2043, 2009.
[19] G. Ghelli, D. Colazzo, and C. Sartiani. Linear time membership in a class of regular expressions with interleaving and counting. In CIKM, pages 389–398, 2008.
[20] S. Grijzenhout and M. Marx. The quality of the XML web. J. Web Sem., 19:59–68, 2013.
[21] D. Hovland. The membership problem for regular expressions with unordered concatenation and numerical constraints. In LATA, pages 313–324, 2012.
[22] E. Kopczyński and A. To. Parikh images of grammars: Complexity and applications. In LICS, pages 80–89, 2010.
[23] W. Martens and F. Neven. On the complexity of typechecking top-down XML transformations. Theor. Comput. Sci., 336(1):153–180, 2005.
[24] W. Martens, F. Neven, and M. Gyssens. Typechecking top-down XML transformations: Fixed input or output schemas. Inf. Comput., 206(7):806–827, 2008.
[25] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for simple regular expressions. In MFCS, pages 889–900, 2004.
[26] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. SIAM J. Comput., 39(4):1486–1530, 2009.
[27] A. J. Mayer and L. J. Stockmeyer. Word problems-this time with interleaving. Inf. Comput., 115(2):293–311, 1994.
[28] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. J. ACM, 51(1):2–45, 2004.
[29] M. Montazerian, P. T. Wood, and S. R. Mousavi. XPath query satisfiability is in PTIME for real-world DTDs. In XSym, pages 17–30, 2007.
[30] F. Neven and T. Schwentick. XML schemas without order, 1999.
[31] F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. Logical Methods in Computer Science, 2(3), 2006.
[32] D. C. O. Oppen. A 2^{2^2^n} upper bound on the complexity of presburger arithmetic. Journal of Computer and System Sciences, 16(3):323–332, 1978.
[33] T. J. Schaefer. The complexity of satisfiability problems. In STOC, pages 216–226, 1978.
[34] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. XMark: A benchmark for XML data management. In VLDB, pages 974–985, 2002.
[35] T. Schwentick. Trees, automata and XML. In PODS, page 222, 2004.
[36] H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In PODS, pages 155–166, 2003.
[37] H. Seidl, T. Schwentick, and A. Muscholl. Counting in trees. In Logic and Automata, pages 575–612, 2008.
[38] S. Staworko and P. Wieczorek. Learning twig and path queries. In ICDT, pages 140–154, 2012.
[39] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In STOC, pages 1–9, 1973.
[40] W3C. XML Path language (XPath) 1.0, 1999.

APPENDIX

A. OMITTED DETAILS

Section 3.2

A. OMITTED DETAILS

Additional notations

Given an atom $A$ (resp. a clause $D$), we denote by $\Sigma_A$ (resp. $\Sigma_D$) the set of symbols occurring in $A$ (resp. $D$). Given a DIME $E$, by $I_E^n$ (resp. $I_E^D$ or $I_E^P$) 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 unambiguous. Moreover, if $E$ is clear from the context, we write simply $I^n$ (resp. $I^A$ or $I^D$) instead of $I^n_E$ (resp. $I^n_E^A$ or $I^n_E^D$). Furthermore, given an interval $I$ which can be either $[n, m]$ or $[n, m]_r$, by $I?$ we understand the interval $[n, m]_r$.

Reduced DIMEs

Recall that an atom has the following grammar:

$$ A := a | a^* | (A^+)^* A. $$

Any DIME can be rewritten as an equivalent reduced DIME, where each clause with interval has one of the following three types:

1. $(A_1 | \ldots | A_k)^+$, where $k \geq 2$ and for any $i \in \{1, \ldots, k\}$ $A_i$ is an atom s.t. there exists $a \in \Sigma_A$ s.t. $I^a = 1$.

   For example, $((a \equiv b^2) | c)^+$ has type 1, but $a^+$ and $(a^* | b^2)^+ | c^+$ do not.

2. $(A_1^1 | \ldots | A_k^k)$, where for any $i \in \{1, \ldots, k\}$ $A_i$ is an atom s.t. there exists $a \in \Sigma_A$ s.t. $I^a = 1$ and $2$ does not belong to the set represented by the interval $I_i$.

   For example, $(a | (b^2 \equiv c^3)^{5, x})$ and $a^+$ have type 2, but $(a \equiv (b^2 \equiv c^3)^{5, x})$ and $(a^* | (b^2 \equiv c^3)^{5, x})$ do not.

3. $(A_1^1 | \ldots | A_k^k)$, where for any $i \in \{1, \ldots, k\}$ $A_i$ is an atom and $I_i$ is an interval s.t. $0$ does not belong to the set represented by the interval $I_i$.

   For example, $(a^* | (b \equiv c)^{3, 4})^+$ and $(a^* \equiv b^2)^+$ have type 3, but $(a^* \equiv b^2)^{3, 4}$ does not.

To obtain reduced DIMEs, we use the following rules:

- Take a simple clause $(A_1^1 | \ldots | A_k^k)$.
  - $(A_1^1 | \ldots | A_k^k)^*$ goes to $(A_1^* | \ldots | A_k^*)$ (k clauses of type 3),
  - $(A_1^1 | \ldots | A_k^k)^+$ goes to $(A_1^* | \ldots | A_k^+)$ (k clauses of type 3) if there exists $i \in \{1, \ldots, k\}$ s.t. $I_i = ?$ or $\Sigma_a = ?$ for any symbol $a \in \Sigma_A$.

- Take a clause $(A_1^1 | \ldots | A_k^k)$.
  - $(A_1^1 | \ldots | A_k^k)^*$ goes to $(A_1^* | \ldots | A_k^*)$ (type 3),
  - $(A_1^1 | \ldots | A_k^k)^+$ goes to $(A_1^* | \ldots | A_k^+)$ (type 3) if there exists $i \in \{1, \ldots, k\}$ s.t. $I_i = ?$ or $\Sigma_a = ?$ for any symbol $a \in \Sigma_A$.

Note that each of the rewriting steps gives an equivalent reduced expression. Form now on, we work with reduced DIMEs only.

Polynomial construction of the characterizing tuple

Before we show how to construct a polynomial representation of the characterizing tuple, we need to introduce a collection of tools that will allow us to represent the set $P_E$ in a compact manner. In fact, the tools are defined for arbitrary languages of unordered words and not necessarily the languages definable with DIMEs. In the remainder we use $L_1$, $L_2$, and $L_3$ to represent arbitrary languages of unordered word, which we shall assume to be nonempty. An empty language can be easily handled as a special case.

We shall work with two classes of positive Boolean combinations of formulas that as atoms use statements of the form $a \in w$. The first class consist of disjunctions only and the second class consist of disjunction of conjunctions (DNF). For instance $a \in w \lor b \in w$ is a formula that belongs to the first class and $(a \in w \land c \in w) \lor b \in w$ is a formula from the second class. We shall represent the formulas from the first class with sets of symbols: $a \in w \lor b \in w$ is represented by $\{a, b\}$, and the formulas from the second class with sets of symbols: $(a \in w \land c \in w) \lor b \in w$ by $\{a, c\}, \{b\}$.

We shall use $X, Y, \ldots$ to range over the representation of the formulas from the first class $X, Y, \ldots$ to range over the representation of the formulas from the second class. The relation of satisfaction of a formula from either class by an unordered word is defined in a straightforward fashion:

$$ w \models X \iff \exists a \in X. a \in w, $$

$$ w \models \chi \iff \exists a \in \chi. \forall a \in A. a \in w. $$

Next, we shall define the theories of a given language i.e., the sets of all formulas from respective classes satisfied by every word of the language:

$$ P_L = \{X \subseteq \Sigma | w \models X\}, \quad P_L = \{\chi \subseteq 2^n | w \models \chi\}. $$

We also define the relation of subsumption between two clauses $X$ and $Y$:

$$ X \subseteq Y \iff \forall A \in X. \exists B \in Y. B \subseteq A, $$

and we say that $X$ subsumes $Y$. We point out that $\subseteq$ is a transitive and reflexive relation (in fact, it is a preor-
Lemma A.1 If $\mathcal{X} \subseteq \mathcal{Y}$, then any word that satisfies $\mathcal{X}$ satisfies also $\mathcal{Y}$.

Next, we establish the relationship between $P_L$ and $\mathcal{P}_L$.

Lemma A.2 $\mathcal{X}$ belongs to $P_L$ iff the set $\{ \{a\} \mid a \in \mathcal{X} \}$ belongs to $\mathcal{P}_L$.

Essentially, we represent the formula $\mathcal{X}$ from the first class as a formula from the second class.

Lemma A.3 $\mathcal{X}$ belongs to $\mathcal{P}_L$ iff any $\mathcal{X} \subseteq \Sigma$ such that $\mathcal{X} \cap A \neq \emptyset$ for every $A \in \mathcal{X}$ belongs to $\mathcal{P}_L$.

Essentially, we distribute the conjunction over disjunction in clauses of the second kind to obtain a set of clauses of the first kind. The two lemmas above allow us to easily establish the following.

Lemma A.4 $P_{L_1} \subseteq P_{L_2}$ iff $\mathcal{P}_{L_1} \subseteq \mathcal{P}_{L_2}$.

Now, we take two sets $\mathcal{R}$ and $\mathcal{S}$ of (representation of) formulas from the second class and say that $\mathcal{R}$ generates $\mathcal{S}$ iff

$$\mathcal{S} = \{ \mathcal{Y} \subseteq \Sigma \mid \exists \mathcal{X}. \mathcal{X} \subseteq \mathcal{Y} \}.$$

We say that a set of formulas $\mathcal{G}$ (of the second kind) is a (theory) generator for $L$ if $\mathcal{G}$ generates $\mathcal{P}_L$. The central result states the instead of working with possibly large theories it is sufficient to reason on their generators alone.

Lemma A.5 Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be theory generators for $L_1$ and $L_2$ respectively. $\mathcal{P}_{L_1} \subseteq \mathcal{P}_{L_2}$ iff every element of $\mathcal{G}_1$ is subsumed by some element of $\mathcal{G}_2$.

Proof. For the if part, take any $\mathcal{X} \in \mathcal{P}_{L_1}$ and let $\mathcal{X}'$ be the element of $\mathcal{G}_1$ that subsumes $\mathcal{X}$ and let $\mathcal{X}''$ be the element of $\mathcal{G}_2$ that subsumes $\mathcal{X}'$. By transitivity of subsumption $\mathcal{X}''$ subsumes $\mathcal{X}$, and therefore $\mathcal{X} \in \mathcal{P}_{L_2}$.

For the only if part, take any $\mathcal{X} \in \mathcal{G}_1$. Naturally, $\mathcal{X} \in \mathcal{P}_{L_1}$, and consequently, $\mathcal{X} \in \mathcal{P}_{L_2}$. Thus, there is $\mathcal{Y} \in \mathcal{G}_2$ that subsumes $\mathcal{X}$.

Consequently, to reason about $P_E = P_{L(E)}$, it suffices to construct a polynomially sized generator, which we present below.

We present the construction for each element of the characterizing tuple and, when necessary, its compact representation. Take a DIME $E = D_1^{i_1} \cdots D_k^{i_k}$.

- The representation of $C_E$ is quadratic in $|\Sigma|$ and can be easily constructed: it consists of all pairs of symbols $(a, b)$ s.t. they appear in different atoms in the same clause of type 2 or 3.
- Recall that the set $N_E$ may be infinite, but it can be represented in a compact manner by using intervals. Given a symbol $a \in \Sigma$, the set $\{i \in \mathbb{N}_0 \mid (a, i) \in N_E\}$ is representable by an interval, denoted $N_E^i(a)$, that can be easily obtained from $E$:

- $N_E^i(a) = [0, 0]$ if $a$ appears in no clause in $E$,
- $N_E^i(a) = [0, \infty]$ (or simply $*$) if $a$ appears in a clause of type 1 in $E$,
- $N_E^i(a) = I^A$ if $I^a = 1$, $A$ is the atom containing $a$, and $A$ is the only atom of a clause of type 2 or 3,
- $N_E^i(a) = I^A'$ 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,
- $N_E^i(a) = [0, \max(I^a)]$ if $I^a = \emptyset$, $A$ is the atom containing $a$, and $A$ belongs to a clause of type 2 or 3.

- Recall that $P_E$ may be exponential in $|\Sigma|$, but we are able to represent it linearly. We essentially represent it using sets of symbols $a$ from clauses of type 1 and 2 s.t. $I^a = 1$, that we denote $G_E^i$ and construct in a straightforward manner:

$$G_E^i = \{ \{a \in S_A \mid I_a = 1\} \mid A \text{ is atom in } D_i \}$$

iff any $u | v$ implies $u$ or $v$. The representation of $K_E$ is quadratic in $|\Sigma|$ and can be easily constructed: it consists of all pairs of distinct symbols $(a, b)$ s.t. they appear in the same atom and $I^a = 1$.

Lemma 3.4 Given an unordered word $w$ and a DIME $E$, $w \in L(E)$ iff $w \models \Delta_E$.

Proof. The only if part follows straightforwardly from the soundness of the polynomial construction of the characterizing tuple $\Delta_E$. For the if part, we take the tuple $\Delta_E$ corresponding to a DIME $E = D_1^{i_1} \cdots D_k^{i_k}$ and an unordered word $w$ s.t. $w \models \Delta_E$. Let $w = w_1 \cdots w_m$, where each $w_i$ contains all occurrences in $w$ of the symbols from $\Sigma_{D_i}$ (for $1 \leq i \leq k$). Since $w \models G_E^i$, we infer that $w^i(a) = 0$ for any $a \in \Sigma \setminus (\Sigma_{D_1} \cup \cdots \cup \Sigma_{D_k})$, which implies that $w^i = \epsilon$. Thus, proving $w \models E$ reduces to proving that for $1 \leq i \leq k$ $w_i \models D_i^{i_k}$. Each derivation can be easily constructed by reasoning on the three possible forms of the $D_i^{i_k}$ (for $1 \leq i \leq k$).

Lemma 3.5 Given two DIMEs $E$ and $E'$, $L(E') \subseteq L(E)$ iff $\Delta_{E'} \preceq \Delta_E$.

Proof. From Lemma 3.4 we know that given a DIME $E$ and an unordered word $w$, $w \in L(E)$ iff $w \models \Delta_E$. Next, we claim that given two DIMEs $E$ and $E'$, $\Delta_{E'} \preceq \Delta_E$ iff $w \models \Delta_{E'}$ implies $w \models \Delta$ for any $w$. The only if part follows directly from the definitions while the if part can be easily shown by contraposition.
Theorem 3.8 Given an unordered word $w$ and two DIMEs $E$ and $E'$:
1. deciding $w \in L(E)$ is in PTIME,
2. deciding $L(E') \subseteq L(E)$ is in PTIME.

Proof. 1) Follows from the polynomial representation of $\Delta_E$. 2) From Lemma 3.5 we know that, given two DIMEs $E$ and $E'$, $L(E') \subseteq L(E)$ iff $\Delta_{E'} \leq \Delta_E$. Note that testing $N_{E'} \subseteq N_E$ is equivalent to testing whether for any symbol $a$, $N_{E'}(a) \subseteq N_E(a)$, which is in PTIME since it reduces to manipulating intervals. From Lemma A.4, Lemma A.5, and the polynomial construction, we infer that testing $P_E \subseteq P_{E'}$ can be done in polynomial time. Finally, we can decide $C_E \subseteq C_{E'}$ and $K_E \subseteq K_{E'}$ in PTIME because each of these sets has a number of elements quadratic in $|\Sigma|$ and can be easily computed in $O(|\Sigma|^2)$.

Section 4.2

Streaming algorithm for membership

We propose a streaming algorithm with earliest rejection for deciding the membership. The algorithm processes an XML document in a single pass, using memory which depends on the height of the tree and not on its size. 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 symbols in $\Sigma$. The input tree $t$ is given in XML format. We assume a well formed stream $\hat{t} \in \Sigma$. The input tree $t$ is given in XML format, and can be easily computed in $O(|\Sigma|^2)$.

Algorithm 1 Streaming algorithm

\begin{algorithm}
\caption{validate($a$)}
\begin{algorithmic}
\State Input: the label of the current node $a \in \Sigma$
\State Output: true if current subtree is valid, false otherwise
\Procedure{validate}{$a$}
\State let $C(N', \mathcal{G}, K)$ be the compact $\Delta_{RS(a)}$
\For{$b \in \Sigma$}
\State count $b = 0$
\State $(\theta, b) = \text{read}(t)$
\State while $\theta = \text{open}$ do
\State count $b := \text{count}[b] + 1$
\If{$\text{count}[b] > \max(N'[b])$}
\State return false
\EndIf
\State if validate($b$) = false then
\State return false
\EndIf
\State if $\exists b \in \Sigma. \text{count}[b] \neq N'(b)$ then
\State return false
\EndIf
\State if $\exists c \in C. (b, c) \in C \land \text{count}[c] 
eq 0$ then
\State return false
\EndIf
\State if $\exists b \in A. \text{count}[b] = 0$ then
\State return false
\EndIf
\State if $\exists (b, c) \in K. \text{count}[b] < \text{count}[c]$ then
\State return false
\EndIf
\State return true
\EndProcedure
\end{algorithmic}
\end{algorithm}

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

\begin{itemize}
\item $r \rightarrow (t_1 \mid f_1) \ldots \mid (t_m \mid f_m)$
\item $t_j \rightarrow c_{j_1} \ldots \mid c_{j_k}$ where $c_{j_1}, \ldots, c_{j_k}$
\item $f_i \rightarrow c_{j_1} \ldots \mid c_{j_k}$ where $c_{j_1}, \ldots, c_{j_k}$
\end{itemize}

The algorithm constructs the compact representation of the characterizing tuple corresponding to its label, which requires space $O(|\Sigma|^2)$. 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 is earliest and rejects a tree as soon as the opening tag is read for nodes that violate either some conflicting pair of siblings or the maximum value for the allowed cardinality.

Proposition 4.5 SAT$_{DIMS,Twig}$ is NP-complete.

Proof. Proposition 4.2.1 from [4] 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:

\begin{itemize}
\item $r \rightarrow (t_1 \mid f_1) \ldots \mid (t_m \mid f_m)$
\item $t_j \rightarrow c_{j_1} \ldots \mid c_{j_k}$ where $c_{j_1}, \ldots, c_{j_k}$
\item $f_i \rightarrow c_{j_1} \ldots \mid c_{j_k}$ where $c_{j_1}, \ldots, c_{j_k}$
\end{itemize}

and the query $q = r[\lceil c_1 \ldots \lceil c_n]$. The formula $\varphi$ is satisfiable iff $(S, q) \in \text{SAT}_{DIMS,Twig}$. The described reduction works in polynomial time in the size of the input formula $\varphi$. Moreover, Theorem 4.4 from [4] implies that satisfiability of twig queries in the presence of DTDs is in NP, which yields the NP upper bound for SAT$_{DIMS,Twig}$.

Proof. Theorem 4.4 from [31] implies that twig query containment in the presence of DTDs is in EXP-TIME. This implies that the problems IMPL$_{DTD,Twig}$, CNT$_{DIMS,Twig}$ are EXP-TIME-complete.

Proof. Theorem 4.4 from [31] implies that twig query containment in the presence of DTDs is in EXP-TIME. This implies that the problems IMPL$_{DTD,Twig}$, CNT$_{DIMS,Twig}$.
IMPL\textsubscript{DIM\textsubscript{S},Twig}, and CNT\textsubscript{DIM\textsubscript{S},Twig} are also in EXP-TIME. The EXPTIME-hardness proof of twig containment in the presence of DTDs (Theorem 4.5 from [31]) has been done using a reduction from Two-player corridor tiling problem and a technique introduced in [28]. In the proof from [31], when testing inclusion \( p \subseteq s, p \) is chosen s.t. it satisfies any tree in \( S \), hence IMPL\textsubscript{DDT,Twig} is also EXPTIME-complete. Furthermore, Lemma 3 in [28] can be adapted to twig queries and DIM\textsubscript{S}: for any \( S \in \text{DIM\textsubscript{S}} \) and twig queries \( q_0, q_1, \ldots, q_m \) there exists \( S' \in \text{DIM\textsubscript{S}} \) and twig queries \( q' \) s.t.

\[
q_0 \subseteq q_1 \cup \ldots \cup q_m \iff q \subseteq q'
\]

Because the DTD in [31] can be captured with DIM\textsubscript{S}, we conclude that IMPL\textsubscript{DIM\textsubscript{S},Twig} and CNT\textsubscript{DIM\textsubscript{S},Twig} are also EXPTIME-complete. \( \Box \)

**Section 4.3**

**Graph simulation**

A simulation of a rooted graph (either dependency graph or universal dependency graph) \( G = (\Sigma, \text{root}, E) \) in a tree \( t \) is a relation \( R \subseteq \Sigma \times N_t \) s.t.

1. \((\text{root}, \text{root}_t) \in R\)
2. for every \((a, n) \in R\), \((a, a') \in E\), there exists \( n' \in N_t \) s.t. \((n, n') \in \text{child}_t\) and \((a', n') \in R\)
3. for every \((a, n) \in R\), \( \text{lab}_t(n) = a \)

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 \( n \in N_t \) s.t. \((a, n) \in R\). If there exists a simulation from \( G \) to \( t \), we write \( t \preceq G \).

The **language of a graph** is \( L(G) = \{ t \in \text{Tree} \mid t \preceq G \} \).

A rooted graph \( G_1 = (\Sigma, \text{root}, E_1) \) is a subgraph of another rooted graph \( G_2 = (\Sigma, \text{root}, E_2) \) if \( E_1 \subseteq E_2 \). For a rooted graph \( G = (\Sigma, \text{root}, E) \), we define the partial order \( \preceq_G \) on the subgraphs of \( G \); given \( G_1 \) and \( G_2 \) two subgraphs of \( G \), \( G_1 \preceq_G G_2 \) if \( G_1 \) is a subgraph of \( G_2 \).

Note that the relation \( \preceq_G \) is reflexive, antisymmetric, and transitive, thus being an order relation. Moreover, it is well-founded and it has a minimal element, that we denote \( G_0 \) for a graph \( G \). Let \( G_0 = (\Sigma, \text{root}, \emptyset) \) and indeed, for any \( G' \) subgraph of \( G \) we have \( G_0 \preceq_G G' \). In the sequel, we assume w.l.o.g. that all subgraphs that we use in our proofs have the property that every edge can be part of a path starting at the root.

**Lemma A.6** For any IMS \( S \), its universal dependency graph can be simulated in any tree \( t \) which belongs to the language of \( S \):

\[
\forall S \in \text{IMS}. \forall t \in L(S). t \preceq G^u_S
\]

**Proof**. Consider an IMS \( S \) and its universal dependency graph \( G^u_S \). Let \( t \) be a tree which belongs to \( L(S) \). We want to construct a witness relation \( R \subseteq \Sigma \times N_t \) for \( t \preceq G^u_S \) and the proof goes by induction on the structure of \( G^u_S \), using the well-founded order \( \preceq_G \) defined above. Let \( P(G) \) denote the statement \( t \preceq G \). Let \( G \) be a subgraph of \( G^u_S \). The induction hypothesis is that for all \( G' \preceq_G G \) and \( G' \neq G \), there exists a relation \( R' \) witness of the simulation \( t \preceq G' \) and we are going to construct \( R \) that witnesses \( t \preceq G \).

For the base case, we take the minimal element for the relation \( \preceq_G \) let it \( G_0 = (\Sigma, \text{root}, \emptyset) \), then \( P(G_0) \) holds for the relation \( R_0 = \{(\text{root}, \text{root}_t)\} \), and therefore, the subgraph containing no edge can be simulated in \( t \).

For the induction case, let \( G \) a subgraph of \( G^u_S \). By the induction hypothesis, we know that \( P(G') \) holds, for every \( G' \preceq_G G \). Consider a subgraph \( G' \) of \( G \) s.t. \( G \) contains exactly one additional edge w.r.t. \( G' \), let the additional edge \((a, a')\) and \( R' \) the witness relation for \( t \preceq G' \). Because \( G' \preceq_G G \) and \((a, a')\) is the only additional edge, we know that \( R' \) already contains images for \( a \) in \( t \) i.e., there exists a node \( n \) s.t. \((a, n) \in R' \).

We construct the relation \( R \) as the union of \( R' \) with \( \{(a', n') \mid \text{lab}_t(n') = a' \land (\exists n. (n, n') \in \text{child}_t \land (a, n) \in R')\} \). The set of tuples that we add is not empty because the edge \((a, a')\) belongs to the universal dependency graph \( G^u_S \), hence for any node labeled by \( a \) in the tree \( t \) there exists a child of it labeled by \( a' \). The construction ensures that \( R \) satisfies all conditions of the definition of a simulation, hence \( t \preceq G \) that yields \( P(G) \) true.

We have proved that \( P(G_0) \) is true and \((\forall G'. G' \preceq_G G \Rightarrow P(G')) \Rightarrow P(G) \), hence \( P(G) \) is true for any \( G \) subgraph of \( G^u_S \), consequently also for \( G^u_S \), hence \( G^u_S \) can be simulated into any tree \( t \) which belongs to the language of \( S \). \( \Box \)

**Graph unfolding**

A **path in a rooted graph** (either dependency graph or universal dependency graph) \( G = (\Sigma, \text{root}, E) \) is a non-empty sequence of vertices starting at root s.t. for any two consecutive vertices in the sequence, there is a directed edge between them in \( G \). By \( \text{Paths}(G) \subseteq \Sigma^* \) 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 Figure 6(b) are \( \text{Paths}(G_1) = \{r, ra, rb, rc, rbd, rcde, rdc\} \).

Similarly, a **path in a tree** \( t \) is a non-empty sequence of nodes starting at root \( t \) s.t. any two consecutive nodes in the sequence are in the relation \( \text{child}_t \). By \( \text{Paths}(t) \subseteq N_t^+ \) we denote the set of all paths in \( t \). Then, we define \( \text{LabPaths}(t) \subseteq \Sigma^+ \) as the set of sequences of labels of nodes from all paths in \( t \). For instance, for the tree \( t_1 \) from Figure 6(a) we have \( \text{Paths}(t_1) = \{n_0, n_0n_1, n_0n_1n_2, n_0n_3, n_0n_3n_4\} \) and \( \text{LabPaths}(t_1) = \{r, ra, rab\} \). Note that \( |\text{LabPaths}(t)| \leq |\text{Paths}(t)| \). The unfolding of a rooted graph \( G = (\Sigma, \text{root}, E) \), denoted \( \text{ug} \), is a tree \( \text{ug} = (\text{ug}, \text{root}_\text{ug}, \text{lab}_\text{ug}, \text{child}_\text{ug}) \), s.t.

- \( \text{ug} = \text{Paths}(G) \),
- \( \text{root}_\text{ug} \in \text{ug} \) is the root of \( \text{ug} \),
\( (p, p.a) \in child\_u_G, \) for all paths \( p, p.a \in Paths(G) \) (note that “@” stands for concatenation),

\( \text{lab}\_u_G(\text{root}\_u_G) = \text{root}, \) and \( \text{lab}\_u_G(p.a) = a, \) for all paths \( p.a \in Paths(G). \)

The unfolding of a graph is finite only when the graph has no cycle reachable from the root, because otherwise \( Paths(G) \) is infinite, so \( u_G \) is infinite. In the sequel we use the unfolding for graphs without any cycle reachable from the root and in this case the unfolding is the smallest tree \( u_G \) (w.r.t. the number of nodes) having \( \text{LabPaths}(u_G) = Paths(G). \) The idea of the unfolding is to transform the rooted 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 Figure 6(b) we take the graph \( G_1 \) and construct its unfolding \( u_{G_1}. \) We remark that the size of the unfolding may be exponential in the size of the graph, for example for the graph \( G_2 \) from Figure 6(c).

\[
\begin{array}{c}
\begin{array}{cccc}
\text{r} & \text{n}_0 & \text{a} & \text{b} \\
\text{n}_1 & \text{a} & \text{b} & \text{c} \\
\text{n}_2 & \text{b} & \text{b} & \text{e} \\
\text{n}_3 & \text{a} & \text{d} & \text{d} \\
\text{n}_4 & \text{d} & \text{e} & \text{e} \\
\end{array}
\end{array}
\]

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

\[
\begin{array}{c}
\begin{array}{cccc}
\text{r} & \text{a}_1 & \text{a}_2 & \text{r} \\
\text{a}_1 & \text{b} & \text{b} & \text{c}_1 \\
\text{a}_2 & \text{b} & \text{b} & \text{c}_2 \\
\text{c}_1 & \text{c}_2 & \text{c}_1 & \text{c}_2 \\
\text{d} & \text{d} & \text{d} & \text{d} \\
\end{array}
\end{array}
\]

(c) Graph \( G_2 \) and its exponential unfolding.

Figure 6: A tree and two graphs with their corresponding unfoldings.

**Extending the definition of embedding**

If a query \( q \) can be embedded in a tree \( t, \) we may write \( t \preceq q \) instead of \( t \models q. \) We also extend the definition of embedding from a query to a tree to 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 \( t \preceq t' \)) if the query \( (N', \text{root}_v', \text{lab}_v', \text{child}_v', \emptyset) \) can be embedded in \( t. \)

Similarly, we can define the embedding from a tree to a rooted graph. Note that two embeddings can be composed, for example:

\( \forall t, t' \in \text{Tree}. \forall q \in \text{Twig}. \quad (t \preceq t' \wedge t' \preceq q \Rightarrow t \preceq q). \)

\( \forall S \in \text{IMS}. \forall t \in \text{Tree}. \forall q \in \text{Twig}. \quad (\text{G}_S^{(n)} \preceq t \wedge t \preceq q \Rightarrow \text{G}_S^{(u)} \preceq q). \)

**Lemma A.7** A rooted graph (dependency graph or universal dependency graph) \( G = (\Sigma, \text{root}, E) \) can be simulated in a tree \( t \) iff its unfolding \( u_G \) can be embedded in \( t. \)

**Proof.** For the *if* part, we know that \( t \preceq u_G \) so there exists a function \( \lambda : N_{u_G} \rightarrow N_t \) which witnesses the embedding of \( u_G \) in \( t. \) We construct a relation \( R \subseteq \Sigma \times N_t \) s.t. \( R = \{(\text{root}, \text{root}_t)\} \cup \{(a, n) \mid \exists p \in N_{u_G}. \ p.a \in N_{u_G} \wedge \lambda(p.a) = n\}. \)

This construction ensures that for every \( (a, n) \in R \) and for every \( (a', a') \in E, \) there exists \( n' \in N_t \) s.t. \( (n, n') \in \text{child}_t \) and \( (a', n') \in R \) because the function \( \lambda \) is a witness for \( t \preceq u_G \) so the child relation is simply translated from \( u_G \) to \( G. \) The construction of \( R \) also guarantees that for every \( (a, n) \in R \) we have \( \text{lab}_t(n) = a \) because \( \lambda \) is the witness for \( t \preceq u_G \) and \( \lambda(p.a) = n. \) Thus we obtain that \( R \) satisfies all conditions to be a simulation of \( G \) in \( t. \)

For the *only if* case, we take a relation \( R \) which witnesses the simulation of \( G \) in \( t. \) We construct the function \( \lambda : N_{u_G} \rightarrow N_t, \) witness of \( t \preceq u_G, \) by recursion on the paths of \( G, \) because \( Paths(G) = N_{u_G}. \) First of all, \( \lambda(\text{root}_u_G) = \text{root}_t. \) We assume that we have a recursive procedure which takes as input a path \( p, \) a symbol \( a, \) and the values of the function \( \lambda \) computed before the procedure call, and it outputs \( \lambda(p.a). \) The invariant of the procedure is that while defining \( \lambda \) for \( p, a, \) \( \lambda \) satisfies the conditions from the definition of embedding for all nodes \( \text{root}_u_G, \ldots, p \) on the path to \( p. \) Furthermore, the values of \( \lambda \) were obtained using the information given by \( R, \) so \( \lambda(p) = n' \) iff \( R(\text{lab}_t(n'), n'). \) Let \( \lambda(p) = n' \) and we construct \( \lambda(p.a) = n, \) where \( R(a, n) \) and \( \text{child}_t(n', n). \) There exists such a node \( n \) because of the recursive construction of \( \lambda \) using \( R \) and the invariant \( \lambda(p.a) = n \) iff \( R(a, n) \) is true. The construction of \( \lambda \) ensures that \( \lambda \) is root-preserving, child-preserving and label-preserving, so it satisfies all conditions to be an embedding from \( u_G \) to \( t, \) so we have found a correct witness for \( t \preceq u_G. \)

**Lemma A.8** A query \( q \) can be embedded in a rooted graph (dependency graph or universal dependency graph) \( G \) iff \( q \) can be embedded in the unfolding tree of \( G. \)

**Proof.** For the *if* part, we know that \( u_G \preceq q, \) so there exists a function \( \lambda : N_q \rightarrow N_{u_G} \) witness of this embedding. We construct a function \( \lambda' : N_q \rightarrow \Sigma \) s.t. \( \lambda'(n) = \text{lab}_{u_G}(\lambda(n)) \) for each node \( n \) from \( N_q. \) Since \( \lambda \) is the witness of the embedding \( u_G \preceq q, \) the constructed \( \lambda' \) satisfies all conditions of the definition of an embedding from \( q \) to \( G. \)
For the only if part, we know that \( G \preceq q \), so there exists a function \( \lambda : N_q \rightarrow \Sigma \) witness of this embedding. We want to construct a function \( \lambda' : N_q \rightarrow N_{u_G} \) to prove \( u_G \preceq q \). We construct \( \lambda' \) by recursion on the tree structure of \( q \). First of all, \( \lambda'(\text{root}_q) = \text{root}_{u_G} \). Then, the induction hypothesis says that \( G \preceq q' \) for any connected subtree \( q' \) obtained from \( q \) by deleting some edges, \( u_G \preceq q' \), which is witnessed by the function \( \lambda' \). Thus, for any node \( n \) of \( q \), \( \lambda'(n) = p \), where \( p \in N_{u_G} \) because \( N_{u_G} = \text{Paths}(G) \) so any node in the unfolding can be identified by a unique sequence of symbols among the paths of \( G \). For the inductive case consider that \( q \) is obtained from \( q' \) by adding one more edge, let it \( (n, n') \). If it is a child edge and \( \lambda'(n) = p \), we construct \( \lambda'(n') = p, \lambda(n') \), which is a path in \( G \) by the definition of the unfolding. Otherwise, if it is a descendant edge and \( \lambda'(n) = p \), we construct \( \lambda'(n') = p, p', \lambda(n') \), where \( p' \) is a randomly chosen path in \( G \) from \( \lambda(n) \) to \( \lambda(n') \). We know by definition of \( \lambda \) that such path exists. The construction ensures that \( u_G \preceq q \), for any \( q \) satisfying the conditions of the recursion, so we can construct a function \( \lambda' \) which is a correct witness for \( u_G \preceq q \).

**Fuse and add operations**

In Figure 7 we present the operations **fuse** and **add**. We say that \( t \preceq_0 t' \) if \( t' \) is obtained from \( t \) by applying one of the operations from Figure 7. 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 any place in the tree. By \( \preceq \) we denote the transitive and reflexive closure of \( \preceq_0 \).

![Figure 7: Operations fuse and add.](image)

Note that the fuse and add operations preserve the embedding i.e., given a twig query \( q \) and two trees \( t \) and \( t' \), if \( t \preceq q \) and \( t \preceq t' \), then \( t' \preceq q \). Furthermore, if we can embed a query \( q \) in a tree \( t \) which can be embedded in the dependency graph of an IMS \( S \), we can perform a sequence of operations s.t. \( t \) is transformed into another tree \( t' \) satisfying \( S \) and \( q \) at the same time. Formally,

**Proposition A.9** Given an IMS \( S \), a query \( q \) and a tree \( t \), if \( G_S \preceq t \) and \( t \preceq q \), 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 \( t \preceq S t' \).

**Family of characteristic graphs**

Given a query \( q \) and a schema \( S \), if \( q \) can be embedded in \( G_S \) then we can capture all trees satisfying \( S \) and \( q \) at the same time with a potentially infinite family of graphs. First, we explain the construction of the characteristic graphs. A characteristic graph \( G \) for a schema \( S \) and a query \( q \) is a tuple \((V_G, \text{root}_G, \text{lab}_G, E_G)\), where \( V_G \) is a finite set of vertices, \( \text{root}_G \in V_G \) is the root of the graph, \( \text{lab}_G : V_G \rightarrow \Sigma \) is a labeling function (with \( \text{lab}_G(\text{root}_G) = \text{root}_S \)), and \( E_G \subseteq V_G \times V_G \) represents the set of edges. Note that for two \( x, y \in \Sigma \cup \{\bullet\} \) we say that \( x \) matches \( y \) if \( y \neq \bullet \) implies \( x = y \). We construct \( G \) with the three steps described below:

1. For any \((n_1, n_2) \in \text{child}_q \), add \( n'_1, n'_2 \) to \( V_G \) and \((n'_1, n'_2) \) to \( E_G \), where \( \text{lab}_G(n'_1) \) matches \( \text{lab}_S(n_1) \) and \( \text{lab}_G(n'_2) \) matches \( \text{lab}_S(n_2) \).
2. For any \((n_1, n_2) \in \text{desc}_q \), choose an acyclic path \( n'_1, \ldots, n'_k \) from \( G_S \), s.t. \( n'_1 \) matches \( \text{lab}_S(n_1) \) and \( n'_k \) matches \( \text{lab}_S(n_2) \). We add to \( G \) the corresponding vertices and edges for this path, as shown for the previous case.
3. For any \( n \in V_G \), take the subgraph from \( G_S^n \) starting at \( \text{lab}_G(n) \) and fuse it in the node \( n \) in the graph \( G \).

In Figure 8(b) we present an example of graph obtained from the embedding from Figure 8(a). We denote by \( G(q, S) \) the set of all graphs obtained from a query \( q \) and a IMS \( S \) using the three steps above, using all embeddings from \( q \) into \( S \). We extend the previous definition of the unfolding to the characteristic graphs. Since a graph \( G \in G(q, S) \) is acyclic, it has a finite unfolding. From the definition it also follows that the size of \( G \) is polynomially bounded by \(|q| \times |S| \) and \( G \preceq q \).

If we allow cyclic paths in step 2, then we obtain similarly the set \( G^*(q, S) \). Note that \( |G(q, S)| \) is finite and may be exponential while \( |G^*(q, S)| \) may be infinite. All trees \( t \in L(S) \cap L(q) \) can be obtained by fuse and add operations (consistently with \( S \)) from the unfolding trees of the graphs in \( G^*(q, S) \):

\[
\forall t \in L(S) \cap L(q), \exists G \in G^*(q, S), u_G \preceq S t
\]

Furthermore, by using a pumping argument, we have:

\[
\forall q \in \text{Twig}, \forall G \in G^*(q, S), (G \preceq q \implies \exists G' \in G(q, S), G' \preceq q).
\]

Given a ME \( E \) and a symbol \( a \), by \( \min_{\text{lab}}(E, a) \) we denote the minimum number of occurrences of the symbol \( a \) in any unordered word defined by \( E \). Next, we define unfolding of a characteristic graph. Given a query \( q \), a IMS \( S \), and a characteristic graph \( G \in G(q, S) \), we construct its unfolding as follows:
Let $u_G$ be the unfolding of $G$ obtained as it has been defined before.

- Update $u_G$ s.t. for any $n \in N_{u_G}$, for any $a \in \Sigma$, 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 $a$.

![Diagram](image)

(a) Embedding $\lambda : N_q \rightarrow G_S$.

(b) Graph $G \in G(q, S)$

**Figure 8: An embedding from a query $q$ to a dependency graph $G_S$ and a graph $G \in G(q, S)$. In $G_S$, the non-nullable edges are drawn with a full line and the nullable edges with a dotted line.**

**Lemma 4.9** Given a twig query $q$ and an IMS $S$:

1. $q$ is satisfiable by $S$ iff $G_S \preceq q$.

2. $q$ is implied by $S$ iff $G_S^* \preceq q$.

**Proof.** (1) For the if part, we know that $G_S \preceq q$, so the family of graphs $G(q, S)$ is not empty. The unfolding of any graph from $G(q, S)$ 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 $G^*(q, S)$. Since $t \preceq q$, we obtain $u_G \preceq q$, so $G \preceq q$, which, from the construction of $G$, implies that $G_S \preceq q$.

(2) For the if part, we know that $G_{S}^* \preceq q$, which implies by Lemma A.7 that $u_{G_{S}^*} \preceq q$. On the other hand, take a tree $t \in L(S)$. By Lemma A.6 we have $t \preceq G_{S}^*$, which implies by Lemma A.7 that $t \preceq u_{G_{S}^*}$. From the last embedding and $u_{G_{S}^*} \preceq q$ we infer that $t \preceq q$. Since $t$ can be any tree in the language of $S$, we conclude that $q$ is implied by $S$.

For the only if part, we know that for any $t \in L(S)$, $t \preceq q$. Consider the tree $t$ obtained as follows: we take $u_{G_{S}^*}$ and we duplicate some subtrees in order to have, for each node $n \in N_t$, $\min_{nb}(R_S(lab_{u_G}(n)), a)$ children labeled by $a$. Naturally, $t$ is in the language of $S$, so $t \preceq q$ from the hypothesis. From the definition of the unfolding, we can infer that $G_{S}^* \preceq t$, which implies that $G_S \preceq q$.

**Theorem 4.11** $\text{CNT}_{IMS,Twig}$ is coNP-complete.

**Proof.** Theorem 4 from [28] implies that $\text{CNT}_{IMS,Twig}$ is coNP-hard. Next, we prove the membership of the problem to coNP. Given an instance $(p, q, S)$, a witness is a function $\lambda : N_p \rightarrow \Sigma$. Testing whether $\lambda$ is an embedding from $p$ to $G_S$ requires polynomial time. If $\lambda$ is an embedding, a non-deterministic polynomial algorithm chooses a graph $G$ from $G(p, S)$ and checks whether $q$ can be embedded in $G$. We claim that $p \not\preceq S q$ iff there exists a $G$ in $G(p, S)$ s.t. $G \preceq q$.

For the if case, we assume that there exists a graph $G \in G(p, S)$ s.t. $G \preceq q$. We know that $G \preceq p$, so $u_G \preceq p$, so there exists a tree $t \in L(S)$ s.t. $t \preceq p$ and $u_G \preceq t$. If we assume by absurd that $t \preceq q$, we have $u_G \preceq q$, so $G \preceq q$, which is a contradiction. We infer thus that there exists a tree $t \in L(S) \cap L(p)$, s.t. $t \not\preceq L(q)$, so $p \not\preceq S q$.

For the only if case, we assume that $p \not\preceq S q$, so there exists a tree $t \in L(S) \cap L(p)$ s.t. $t \not\preceq L(q)$. Because $t \in L(S) \cap L(p)$, we know that there exists a graph $G \in G^*(p, S)$, s.t. $u_G \preceq t$. We know that $t \not\preceq q$, so $u_G \not\preceq q$, so $G \not\preceq q$. Moreover, we know using the pumping argument that in this case there exists a graph $G^* \in G(p, S)$ s.t. $G^* \not\preceq q$.

**Extending the complexity results to disjunction-free DTDs**

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, essentially regular expressions of the form:

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

where $a \in \Sigma$. Given such an expression $E$, consider the set $\text{non\_nullable}(E)$ which contains the symbols present in all words from $L(E)$. Formally,

$$\text{non\_nullable}(E) = \{a \in \Sigma | \forall w \in L(E). \exists w_1, w_2, w = w_1 \cdot a \cdot w_2\}.$$
We can compute $\text{non Nullable}(E)$ recursively:
\[
\text{non Nullable}(\epsilon) = \text{non Nullable}(E^* ?) = \emptyset,
\]
\[
\text{non Nullable}(a) = \{a\},
\]
\[
\text{non Nullable}(E_1 \cdot E_2) = \text{non Nullable}(E_1) \cup \text{non Nullable}(E_2),
\]
\[
\text{non Nullable}(E^+) = \text{non Nullable}(E).
\]

Similarly, let $\text{Nullable}(E)$ the set containing symbols which appear in at least one word from $L(E)$. Formally,
\[
\text{Nullable}(E) = \{a \in \Sigma \mid \exists w \in L(E). \exists w_1, w_2. w = w_1 \cdot a \cdot w_2\}.
\]

We can compute $\text{Nullable}(E)$ recursively:
\[
\text{Nullable}(\epsilon) = \emptyset,
\]
\[
\text{Nullable}(a) = \{a\},
\]
\[
\text{Nullable}(E^+ ?) = \text{Nullable}(E),
\]
\[
\text{Nullable}(E_1 \cdot E_2) = \text{Nullable}(E_1) \cup \text{Nullable}(E_2).
\]

Next, we adapt the notions of dependency graph and universal dependency graph for disjunction-free DTDs.

The dependency graph of a disjunction-free DTD $S$ is a directed rooted graph $G_S = (\Sigma, \text{root}_S, E_S)$, where
\[
E_S = \{(a, a') \mid a' \in \text{Nullable}(R_S(a))\}.
\]

Similarly, the universal dependency graph of a disjunction-free DTD $S$ is a directed rooted graph $G^\text{universal}_S = (\Sigma, \text{root}_S, E^\text{universal}_S)$, where
\[
E^\text{universal}_S = \{(a, a') \mid a' \in \text{non Nullable}(R_S(a))\}.
\]

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 does not exist any tree consistent with the schema and containing any of the symbols implied in that cycle.

For a symbol $a \in \Sigma$ and a disjunction-free regular expression $E$, by $\text{min nb}(E, a)$ we denote the minimum number of occurrences of the symbol $a$ in any word defined by $E$.

\[
\text{min nb}(\epsilon, a) = \text{min nb}(E^*, a) = \text{min nb}(E^?, a) = 0,
\]
\[
\text{min nb}(a, a) = 1,
\]
\[
\text{min nb}(E_1 \cdot E_2, a) = \text{min nb}(E_1, a) + \text{min nb}(E_2, a),
\]
\[
\text{min nb}(E^+, a) = \text{min nb}(E, a).
\]

**Theorem 4.12** \text{IMPL}_{\text{disj-free-DTD,Twig}} is in \text{PTIME} and \text{CNT}_{\text{disj-free-DTD,Twig}} is \text{coNP-complete}.

**Proof.** We claim that a query $q$ is implied by a disjunction-free DTD $S$ iff $G^q_S \preceq q$ and since the embedding of a query in a graph can be computed in polynomial time, this implies that \text{IMPL}_{\text{disj-free-DTD,Twig}} is in \text{PTIME}. The proof follows immediately from the proof of Lemma 4.9(2). Theorem 4 from [28] implies that CNT $\text{disj-free-DTD,Twig}$ is coNP-hard. The membership of CNT $\text{disj-free-DTD,Twig}$ to coNP follows from the proof of Theorem 4.11. The two proofs can be adapted because given a disjunction-free regular expression $E$, there exists a word $u \in L(E)$ which is an ordering of the multiset $w = \bigcup_{a \in \Sigma} a^{\text{min nb}(E, a)}$. Moreover, the order imposed by the DTD on the elements is not important because we work with twig queries which are order-oblivious. \qed