Efficient enumeration for conjunctive queries over x-underbar structures

Guillaume Bagan, Arnaud Durand, Emmanuel Filiot, Olivier Gauwin

Institutions: French Institute for Research in Computer Science and Automation, Paris Diderot University, Université libre de Bruxelles, University of Mons

Published on: 23 Aug 2010 - Computer Science Logic

Topics: Conjunctive query, Boolean conjunctive query and Time complexity

Efficient Enumeration for Conjunctive Queries over X-underbar Structures

Guillaume Bagan, Arnaud Durand, Emmanuel Filiot, Olivier Gauwin

Efficient Enumeration for Conjunctive Queries over X-underbar Structures

Guillaume Bagan[1], Arnaud Durand[2], Emmanuel Filiot[3], and Olivier Gauwin[4]

[1] INRIA Lille – Mostrare
[2] ELM, CNRS FRE 3233 – Université Paris Diderot
[3] Université Libre de Bruxelles
[4] Université de Mons – UMONS

Abstract. We investigate efficient enumeration algorithms for conjunctive queries for databases over binary relations that satisfy the $\underline{\text{X}}$ property. Tree-like relations such as XPath axes or grids are natural examples of such relations. We first show that the result of an n-ary conjunctive query Q over an $\underline{\text{X}}$ structure S can be enumerated with a delay in $O(n\cdot|S|\cdot|Q|)$ between two consecutive n-tuples. Then, we consider acyclic conjunctive queries and show that such queries admit an enumeration algorithm with delay $O(|Q|\cdot|D|)$ and a preprocessing in $O(|Q|\cdot|S|)$ where D is the domain of S. The delay can even be improved to $O(n{\cdot}|D|)$ with a slightly more expensive preprocessing step. As an application of our method, we also show that any n-ary XPath acyclic conjunctive query Q over an unranked tree t can be enumerated with a preprocessing and delay $O(|Q|\cdot|t|)$. In the second part of the paper, we consider conjunctive queries with possible inequalities ($\neq$) between variables. In this case, we show that query evaluation is NP-hard and, unless $\text{P} = \text{NP}$, these queries do not admit enumeration algorithms with a combined polynomial time delay. However, we also show that hardness relies only on the number ℓ of variables that appear in inequalities. We propose efficient enumeration procedures for acyclic and general conjunctive queries whose delay is exponential in ℓ but polynomial (even quasi-linear) in $|Q|$ and $|S|$.

1 Introduction

Querying is a core task in database processing. Given a relational structure S, an n-ary query retrieves n-tuples of elements in the domain of S. For XML databases, the structure S is a tree modeling an XML document, and an n-ary query selects n-tuples of nodes of this tree.

Query computing can be seen as a generation process. In this case, one tries to output selected tuples one after the other, without duplicates, while minimizing the delay between successive answers. Good enumeration algorithms then carry additional information on the problem: it shows that the query is not only tractable but that the result can be obtained step by step in a very regular way. A nice feature is that enumeration permits to start a pipeline process, and thus allow tasks based on querying to start processing answers without waiting for the whole evaluation to be done.

More precisely, an enumeration algorithm is the process of generating one after the other the tuples of a given query $Q(S)$ for some given order $<$ [4]. In particular, the enumeration process does not generate duplicates. The so-called *preprocessing* is a preliminary step performed before the enumeration phase. The *delay* of such an algorithm is the maximum between (i) the time to get the first tuple after preprocessing (ii) the maximal interval of time between the generation of two consecutive solutions for $<$.

In [13], a tractability frontier is established for conjunctive queries defined using XPath axes as predicates. This dichotomy relies on the $\underline{\mathrm{X}}$ property (see Section 2 for a definition): if a structure S has the $\underline{\mathrm{X}}$ property w.r.t. some total order $<$ on the domain D of the structure, then it can be checked in PTIME combined complexity whether the result of a conjunctive query Q over S is empty or not. If Q is an n-ary query, then the proof provides an evaluation algorithm with time complexity $O(|D|^n \cdot |S| \cdot |Q|)$. This holds for any conjunctive queries using unary and binary predicates, not only XPath expressions. For all conjunctive queries based on XPath axes combinations that do not have the $\underline{\mathrm{X}}$ property, checking whether Q selects some tuples on S is NP-complete.

Based on the aforementioned dichotomy, we investigate enumeration of conjunctive queries over $\underline{\mathrm{X}}$ structures, by providing enumeration algorithms (see Fig. 1) and hardness results. Enumeration complexity of query problems, in particular on tree structures and on tree-like queries, has deserved some attention recently [3, 7, 4]. However grids also have the $\underline{\mathrm{X}}$ property, hence new tools are needed for such kind of structures.

Prior to this investigation, we show that deciding whether a structure S is $\underline{\mathrm{X}}$ for some order on its domain is NP-hard when S contains two binary relations. If S contains only one binary relation, the problem is known to be in PTIME. [5]

Our first algorithm enumerates answers of an n-ary conjunctive query Q over an $\underline{\mathrm{X}}$ structure S without preprocessing and with a delay in $O(n \cdot |S| \cdot |Q|)$. The method relies on the computation of maximal arc-consistent pre-valuations proposed in [13]. We then turn to *acyclic* n-ary conjunctive queries (ACQS) Q over $\underline{\mathrm{X}}$ structures S, and propose an algorithm with preprocessing $O(|Q| \cdot |S|)$ and delay in $O(|Q| \cdot |D|)$, where D is the domain of S. We first reason on a conceptually simpler notion of tree-shaped queries, namely tree patterns. ACQS are then mapped to tree patterns. With a slightly more expensive preprocessing in $O(|Q| \cdot |D|^2)$, this delay can be reduced to $O(n \cdot |D|)$. This is to be compared with Yannakakis' algorithm [18] turned into an enumeration algorithm in [4], which computes all answers $Q(S)$ of an acyclic conjunctive query Q over an arbitrary structure S with delay $O(|Q| \cdot |S|)$, hence in total time $O(|Q| \cdot |S| \cdot |Q(S)|)$.

The set of XPath axes does not have the $\underline{\mathrm{X}}$ property (only some combinations of them have it). However we show that the main ideas of the enumeration algorithm for ACQS over $\underline{\mathrm{X}}$ can also be applied to ACQS over XPath axes. Given an unranked tree t and an ACQ Q over XPath axes, $Q(t)$ can be enumerated with a preprocessing and delay $O(|Q| \cdot |t|)$, where $|t|$ is the number of nodes. This is particularly relevant to XML processing as XPath has become one of the main query language for XML. Moreover, as XML documents can be very large in practice, even a square in the size of the tree for preprocessing and delay would not be feasible.

It is a natural question to ask whether the delay can be made polynomial in the size of the query only (and independent of the size of the $\underline{\mathrm{X}}$ structure) in the case of conjunctive or even acyclic conjunctive queries. While this is still an open question, we obtain hardness results and tight algorithms when allowing inequalities in predicates.

[5] personal communication with Pavol Hell and Arash Rafiey. To be published.

Source	Queries	Structures	Preprocessing	Delay
[13]	CQ	$\underline{X}$	$O(1)$	$O(\|D\|^n \cdot \|Q\| \cdot \|S\|)$
Section 3	CQ	$\underline{X}$	$O(1)$	$O(n \cdot \|Q\| \cdot \|S\|)$
Section 5	CQ($\neq$)	$\underline{X}$	$O(1)$	$O(\ell^{O(\ell)} \cdot \|Q\| \cdot n \cdot \|S\| \cdot \log \|D\|)$
[4]	ACQ	all	$O(1)$	$O(\|Q\| \cdot \|S\|)$
Section 4	ACQ	$\underline{X}$	$O(\|Q\| \cdot \|S\|)$	$O(\|Q\| \cdot \|D\|)$
Section 4	ACQ	$\underline{X}$	$O(\|Q\| \cdot \|D\|^2)$	$O(n \cdot \|D\|)$
Section 4	ACQ (XPath axes)	tree t	$O(\|Q\| \cdot \|t\|)$	$O(\|Q\| \cdot \|t\|)$
Section 5	ACQ ($\neq$)	$\underline{X}$	$O(\|Q\| \cdot \|S\|)$	$O(\ell^{O(\ell)} \cdot \|Q\| \cdot \|D\| \cdot \log \|D\|)$

Fig. 1. Enumeration algorithms for n-ary queries Q over structures S with domain D. ℓ is the number of variables used in inequalities, while $|t|$ denotes the number of nodes of the tree t.

For conjunctive queries with inequalities over $\underline{X}$ structures, satisfiability is NP-hard, even when the query restricted to $\underline{X}$ predicates is acyclic. As a consequence, these queries may not be enumerated with a polynomial delay, in terms of combined complexity. However, we propose an enumeration algorithm for such queries without preprocessing, and with a delay in $O(\ell^{O(\ell)} \cdot |Q| \cdot n \cdot |S| \cdot \log |D|)$ where the blowup is only in the number ℓ of distinct variables in inequalities. In the acyclic case, when allowing a preprocessing phase in $O(|Q| \cdot |S|)$, we obtain a delay in $O(\ell^{O(\ell)} \cdot |Q| \cdot |D| \cdot \log |D|)$.

Related work. Enumeration of acyclic conjunctive queries has been studied in [4]. It is shown that the result of such a query can be enumerated with a delay linear in $|S| \cdot |Q|$ (dependency on $|Q|$ is exponential when inequalities are allowed). They also formulate conditions on free variables to have a delay depending on $|Q|$ only. Monadic second order queries on structures of bounded treewidth are considered in [7, 3] where enumeration algorithms are exhibited with a delay depending on $|Q|$ (non elementarily) and on the size of each tuple.

For acyclic conjunctive queries, Koch [16] provides an enumeration algorithm with preprocessing in $O(|S| \cdot |Q|)$ (for computing Θ) and delay $O(|D|)$, where $|D|$ is the domain size. For tree structures and tree-like queries over child/descendant axes, the delay can be improved to $O(1)$ with the algorithm *TwigStack* [5]. However in both works, all variables are considered as free. In our work, variables can be existentially quantified, so that it may be an exponential number of valuations (in the size of the domain) for a single answer tuple. However we can still achieve a polynomial delay between two successive tuples.

Conjunctive queries over child/descendant axes are considered in [6] for the class of graphs. No enumeration algorithms are provided, but query evaluation algorithms. In the case of tree-like queries over graphs the time complexity is $O(|Q| \cdot |D| \cdot |E| + |Q(S)|)$, where $|V|$ is the size of the domain and $|E|$ is the number of edges in the graph. However it is exponential in the number of variables in the case of graph queries. In [9], XPath dialects that can define n-ary queries are introduced. No enumeration algorithm is given, but a fragment that corresponds to union of ACQs over trees t is defined, for which evaluation is in time $O(n \cdot |Q| \cdot |t|^2 \cdot |Q(t)|)$.

2 $\underline{X}$ Structures

We consider relational structures over binary relational symbols only[6]. Formally let σ be a signature $\{R_1, \ldots, R_m\}$ of binary relational symbols R_i (equality $=$ is part of the language too). A (finite) σ-structure S consists of a finite domain D together with an interpretation of σ-symbols R_i as binary relations R_i^S on D (when it is clear from the context we do not distinguish a symbol from its interpretation). Let $\mathcal{R}$ be the set $\{R_1^S, \ldots, R_m^S\}$. The size $|D|$ of a domain D is its cardinality. The size of a σ-structure S over D, is $|S| = |D| + \sum_{R\in\mathcal{R}} |R|$, where $|R| = |\{(v, v') \in D \times D \mid R(v, v')\}|$ Given two binary relations R_1 and R_2 of $\mathcal{R}$, one defines the relations $R_1 \circ R_2$, $R_1 \cap R_2$, $R_1 \cup R_2$ and R_1^{-1} for the composition, intersection, union and inverse in the standard way. Given a set $A \subseteq D$, we denote by $R(A)$ the set $\{b \mid \exists a \in A.\ R(a, b)\}$.

The computation model used in this paper is a $\{+\}$-RAM with uniform cost measure as in [3, 4]. It takes σ-structure as input (with each tuple in a distinct input register) and uses, during the computation, register contents and addresses always bounded by $O(D)$ (hence the correspondence with logarithmic cost is immediate).

The $\underline{X}$ property [14]. A binary relation R over D has the *$\underline{X}$ property* w.r.t. a total order $<$ on D iff for all elements v_0, v_1, v_2, v_3 of D the following holds:

$$(\underline{X}\text{ property}) \qquad R(v_0, v_1) \wedge R(v_2, v_3) \implies R(\min(v_0, v_2), \min(v_1, v_3))$$

A binary relation having the $\underline{X}$ property is also called an *$\underline{X}$ relation*. We say that a set of binary relations $\mathcal{R}$ over D has the $\underline{X}$ property if there is a total order $<$ on D such that all relations of $\mathcal{R}$ have the $\underline{X}$ property w.r.t. $<$. Similarly, a structure has the $\underline{X}$ property if its relations have the $\underline{X}$ property. We call it an $\underline{X}$ structure.

Example 1. Over tree structures, XPath axes define classical binary relations such as child, parent, descendant, ancestor, next-sibling, etc. There is no order $<$ on the set of nodes such that all XPath axes are $\underline{X}$ w.r.t. to the same $<$. However, such orders exist when considering some subsets of axes. For instance, $\{\text{child}, \text{next-sibling}\}$ are $\underline{X}$ for the order induced by a breadth-first left to right traversal of the tree. A complete list of subsets of tree relations (XPath axes) having the $\underline{X}$ property is established in [13].

Example 2. The $n \times n$-grid graph $G = (V, E)$ with $V = \{1, ..., n\}^2$ and for all $i, i', j, j' \in \{1, ..., n\}$, $((i, j), (i', j')) \in E$ if and only if $\{|i - i'|, |j - j'|\} = \{0, 1\}$, is $\underline{X}$ for the lexicographic extension of the natural ordering $<$ on $\{1, ..., n\}$.

Lemma 1. *The class of $\underline{X}$ relations is closed by composition, intersection, and inverse (even for the same order). However, it is not closed by union and complement (even for different orders).*

Proof. Closure by composition, intersection and inverse are easy to check. For union, suppose that $v_0 < v_1$ and consider the relations $R_1 = \{(v_0, v_1)\}$ and $R_2 = \{(v_1, v_0)\}$. Both have the $\underline{X}$ property w.r.t. $<$, but not $R_1 \cup R_2$ (for both linear orders on v_0, v_1). For complement, consider the identity relation $\{(v_0, v_0), (v_1, v_1)\}$, its complement is exactly $R_1 \cup R_2$. $\square$

[6] To ease notations, we do not consider unary relation symbols, but all the results of the paper carry over to a signature with both binary and unary relation symbols.

Sorted representation of $\underline{\mathrm{X}}$ relations. In this paper, for $\underline{\mathrm{X}}$ structures, we assume that the order $<$ is given and that comparison can be done in $O(1)$. Relations are given by sets of pairs of elements and the domain D is given as a list of elements sorted according to $<$. To perform some operations more efficiently, we use the following so-called *sorted representation* for a relation structure S. Every element u of D is represented by an integer $i_u \in \{1, \dots, |D|\}$. Moreover, we require that $i_u <_{\mathbb{N}} i_v$ iff $u < v$. Every relation R is represented by two arrays A and A^{-1} of size $|D|$ such that for all $i_u \in \{1, \dots, |D|\}$, $A[u]$ is the sorted (increasing) list (for $<$) of successors of u in R, and $A^{-1}[u]$ is the increasing list of successors of u in R^{-1}. In other words, A is an adjacency list representation of R, viewed as a directed graph with vertex set D. We also require that the list is doubly-linked, so that we can traverse the list in both orders.

Lemma 2. *For every structure S with some total order $<$ on its domain D, whose domain and relations are represented by a sorted list and lists of pairs of elements respectively, one can compute a sorted representation of S in time $O(|S|)$.*

Proof. One first renames the elements of D and S into integers. We do it such that for all $i_u, i_v \in \{1, \dots, |D|\}$, $i_u <_{\mathbb{N}} i_v$ iff $u < v$. This can be done in $O(|S|)$ since D is assumed to be given as a sorted list of elements (for $<$). Then for each relation R, one has to construct two arrays A and A^{-1} of size $|D|$ such that for all $u \in D$, $A[i_u]$ is the sorted list of successors of u in R. It is known that A can be computed in $O(|R|)$ by applying two times the following algorithm: for all i_u from 1 to $|D|$ and for all i_v such that $R(u, v)$, append i_u to $A[i_v]$. Applying this algorithm a second time on A results in sorted lists of successors for each u. This is done in $O(|R|)$. One can compute A^{-1} similarly. $\square$

Given a subset $A \subseteq D$, we show that, for $\underline{\mathrm{X}}$ structures, this representation allows us to compute $R(A)$ and $R^{-1}(A)$ efficiently.

Lemma 3. *For every $\underline{\mathrm{X}}$ relation R over a set D in sorted representation, and every set $A \subseteq D$ given as a list, $R(A)$ and $R^{-1}(A)$ can be computed in time $O(|D|)$.*

Proof. See Algorithm 1. The set A can be sorted in $O(|D|)$ as we know that there are at most $|D|$ integers in A with maximal value $|D|$. The assumed orders on v and w elements ensure that each element $v \in A$ and $w \in R(A)$ is processed only once. The $\underline{\mathrm{X}}$ property allows to skip w elements lower than w elements already processed. Note that the algorithm computes a sorted set. Since R^{-1} is also $\underline{\mathrm{X}}$ for $<$, one can apply the same algorithm on the representation of R^{-1}. $\square$

Thanks to Lemma 3, we obtain the following proposition:

Proposition 1. *The composition of two binary relations over D can be computed in time $O(|D|^2)$, whenever one of them is $\underline{\mathrm{X}}$. In other terms, the product of two $n \times n$ Boolean matrices is computable in linear time when one of them satisfies the $\underline{\mathrm{X}}$ property.*

Proof. One first computes a sorted representation in $O(|R_1|+|R_2|) = O(|D|^2)$ (Lemma 2). For $R_1 \circ R_2$, $R_1(R_2(v))$ (for $v \in D$) can be computed in time $O(|D|)$ according to Lemma 3, so $R_1 \circ R_2$ can be computed in time $O(|D|^2)$. For $R_2 \circ R_1$, we have: $R_2 \circ R_1 = (R_1^{-1} \circ R_2^{-1})^{-1}$. Inverting a relation can be done in $O(|D|^2)$, and $\underline{\mathrm{X}}$ relations are closed under inverse (Lemma 1). $\square$

Algorithm 1 Computing $R(A)$

```
procedure IMAGE($R, A$)
    sort $A$
    $S \leftarrow$ empty list;    $max \leftarrow -\infty$
    for $v \in A$ w.r.t. $<$ increasing do
        for $w \in R(v)$ w.r.t. $<$ decreasing do
            if $w \leq max$ then
                exit inner for-loop
            $S.append(w)$
        $max \leftarrow \max(R(v))$
    sort $S$
    return $S$
```

Finding $\underline{X}$ orders. In this part, one considers the problem of checking whether, for a given relation R, there exists an order for which R satisfies the $\underline{X}$ property. This problem has been considered under different angles in the CSP literature (see for example [15]).

$\underline{X}$-ENUMERATION(n)
Input: a finite domain D and $R_1, \ldots, R_n \subseteq D^2$
Question: does there exist a common $\underline{X}$-enumeration for $R_1, \ldots, R_n$ (*i.e.* a total order $<$ on D such that $R_1, \ldots, R_n$ are $\underline{X}$ for $<$)?

We prove that deciding whether two relations are $\underline{X}$ for some total order is NP-complete. Hardness is proved by reduction from BETWEENNESS [17, 11].

Proposition 2. *The problem* $\underline{X}$-ENUMERATION(2) *is* NP*-complete.*

Proof. Given a linear order, one can check in polynomial time whether the two relations are $\underline{X}$ with respect to this order, thus proving easiness. The hardness is proved by reduction from the NP-complete problem named BETWEENNESS [17, 11].

BETWEENNESS
Input: a finite set A and a collection I of ordered triples (a, b, c) of distinct elements from A
Question: is there a betweenness ordering f of A for I, that is, a one-to-one function $f : A \rightarrow \{1, 2, ..., |A|\}$ such that for each (a, b, c), either $f(a) < f(b) < f(c)$ or $f(c) < f(b) < f(a)$?

Let A be a finite set and $I \subseteq A^3$. Let $|I| = m$. One constructs m copies $A_1, ..., A_m$ of A (with $A = A_1$) and two relations S and R as follows.

- relation S is (the graph of) a bijection from A_i to A_{i+1} for $i < m$.
- let $t_i = (a, b, c)$ be the ith triple of I (for some arbitrary enumeration of the triples), then construct $R(a_i, b_i)$ and $R(b_i, c_i)$ where a_i, b_i, c_i belong to A_i and are the unique elements related to, respectively, a, b and c by a S-path of length $i - 1$.

Let $\prec$ be a linear ordering of elements of $\bigcup_{i=1}^{n} A_i$ and suppose that S and R are $\underline{\mathrm{X}}$ for $\prec$. By construction of relation S, for each $i < m$ and all x_i, y_i in A_i, their images x_{i+1} and y_{i+1}, by S, are such that :

$$x_i \prec y_i \iff x_{i+1} \prec y_{i+1}. \tag{1}$$

If not, suppose we have $x_i \prec y_i$ and $x_{i+1} \succ y_{i+1}$ then, since $S(x_i, x_{i+1})$ and $S(y_i, y_{i+1})$ holds but not $S(x_i, y_{i+1})$, relation S would not be $\underline{\mathrm{X}}$ for $\prec$. Similarly, for $x_i \succ y_i$ and $x_{i+1} \prec y_{i+1}$ the same conclusion holds. It follows that $\prec$ preserves the ordering of A in its various copies (note however that this does not imply that elements of distinct copies always compare the same *i.e.* that $A_i \prec A_{i+1}$). Now, an easy calculation shows that in each copy A_i of A, since relation R is $\underline{\mathrm{X}}$ for $\prec$ then $a_i \prec b_i \prec c_i$ or $c_i \prec b_i \prec a_i$. Hence, the betweenness property holds for I and the successor function associated to $\prec$ on A.

For the converse, suppose that I satisfies the betweenness property for some function f. Let $\prec$ be the linear order that extends the successor f. Now extend $\prec$ on all copies of A such that Equivalence 1 is preserved. Assume also that the copies are ordered in the following way : $A_1 \prec A_2 \prec \ldots \prec A_n$. In that case, it is easily checked that S is $\underline{\mathrm{X}}$ for $\prec$. Also, since for all triples $t_i = (a, b, c)$, it holds that $f(a) < f(b) < f(c)$ or $f(c) < f(b) < f(a)$, it also holds that $a_i \prec b_i \prec c_i$ or $c_i \prec b_i \prec a_i$. Hence, again by easy calculation, R is $\underline{\mathrm{X}}$. □

For the case of one binary reflexive relation, it has been shown [8] that one can check in polynomial time whether R has an $\underline{\mathrm{X}}$ enumeration. Recently, Hell and Rafiey (personal communication) proved that the problem $\underline{\mathrm{X}}$-ENUMERATION(1) is in P.

3 Conjunctive queries over $\underline{\mathrm{X}}$ structures

Queries. An n-ary *query* Q over a structure $S = (D, \mathcal{R})$ is a mapping from S to 2^{D^n}. The set $Q(S)$ is also called the *answer set* over S. Conjunctive queries are defined in the normal way [1]. In particular, an n-ary *conjunctive query* over S is a query defined by an existential first-order formula without negation nor disjunction, with n free variables and using relations from $\mathcal{R}$ as predicates. A 0-ary conjunctive query is called a *Boolean* conjunctive query. We recall that all the relations considered in this paper are binary. We write vars(Q) for the variables occurring in Q, and vars$_{\text{free}}(Q)$ for the n free ones. We also write $R(x, y) \in Q$ if $R(x, y)$ appears in Q. Throughout this paper, we assume that formulas defining conjunctive queries are in prenex normal form. The *body* of Q is obtained from Q by removing its quantifiers. The size of a conjunctive query Q, denoted by $|Q|$, is the number of symbols of its first-order formula.

Pre-valuations and valuations. Given a conjunctive query Q over a structure $S = (D, \mathcal{R})$, we say that Θ is a *pre-valuation* for Q if it is a total function Θ : vars$(Q) \to 2^D$ assigning a nonempty set of elements of D to each variable of Q. A pre-valuation Θ is *arc-consistent* on S iff for each binary predicate $R(x, y)$ of Q, for each $v \in \Theta(x)$, $R(v, w)$ is true for some $w \in \Theta(y)$, and for each $w \in \Theta(y)$, $R(v, w)$ is true for some $v \in \Theta(x)$.

A *valuation* θ is a total function $\theta : \text{vars}(Q) \to D$ assigning an element of D to each variable of Q. A valuation is *consistent* if it satisfies the body of Q. Conjunctive queries define n-ary queries in the following sense: $Q(S)$ is the set of tuples $(\theta(x_1), \ldots, \theta(x_n))$ such that θ satisfies Q and $\text{vars}_{\text{free}}(Q) = \{x_1, \ldots, x_n\}$. The *minimum* valuation θ in Θ w.r.t. some total order $<$ on D is written $\min_< \Theta$ and given by: $\theta(x) = \min_< \Theta(x)$ for all $x \in \text{vars}(Q)$. Valuations are ordered according to the lexicographical extension of $<$. The following properties will be the basis of our enumeration algorithm.

Lemma 4 (Gottlob, Koch, Schulz [13]). *Let S be a structure and Q a conjunctive query on S.*

1. *the unique subset-maximal arc-consistent pre-valuation of Q on S can be computed in time $O(|S| \cdot |Q|)$.*
2. *if all the relations in S are $\underline{X}$ w.r.t. the same order $<$, then for any arc-consistent pre-valuation of Q on S, the corresponding minimum valuation is consistent.*

This lemma provides a procedure to decide whether $Q(S) = \emptyset$ for every n-ary conjunctive query Q over an $\underline{X}$ structure S, in time $O(|S| \cdot |Q|)$. It suffices to compute the subset-maximal arc-consistent pre-valuation Θ of Q on S, and check that $\Theta(x) \neq \emptyset$ for all $x \in \text{vars}_{\text{free}}(Q)$. Equivalently, when Q has no free variable (*i.e.* $n = 0$), evaluating Q on S can be done in time $O(|S| \cdot |Q|)$.

In [13], a first evaluation algorithm is proposed for n-ary queries $Q(x_1, \ldots, x_n)$ over $\underline{X}$ structures. It consists in enumerating all tuples $(u_1, \ldots, u_n) \in D^n$, and for each of them, check the satisfiability of $Q(u_1, \ldots, u_n)$ where free variables are interpreted by $u_1, \ldots, u_n$ respectively. This algorithm outputs the answers of Q on S in time $O(|D|^n \cdot |S| \cdot |Q|)$. However the delay may be $O(|D|^n \cdot |S| \cdot |Q|)$.

In this section, we explain how to extend this algorithm into an enumeration algorithm without preprocessing, and a delay in $O(n \cdot |S| \cdot |Q|)$. The core idea is to consider distinct domains $D_1, \ldots, D_n$ for the free variables $x_1, \ldots, x_n$ of Q. This allows us to update these domains, in order to avoid duplicate answers and to ensure the enumeration of all answers in lexicographical order w.r.t. $<$.

In the sequel we will consider arc-consistent pre-valuations for S restricted to domains defined by $\overline{D} = (D_1, \ldots, D_n)$, with $D_i \subseteq D$ for all $1 \leq i \leq n$. To define this formally, we introduce fresh unary relation symbols $\widetilde{D_i}$, and consider the signature $\sigma' = \sigma \uplus \widetilde{D_1} \uplus \ldots \uplus \widetilde{D_n}$. Consider the query $Q' = Q \wedge \widetilde{D_1}(x_1) \wedge \ldots \wedge \widetilde{D_n}(x_n)$ and the σ'-structure S' that is similar to S, but extends it by interpreting $\widetilde{D_i}$ in the following way: $\widetilde{D_i}^{S'} = D_i$. Then we define $pv_{max}(Q, S, \overline{D})$ as the unique subset-maximal arc-consistent pre-valuation for Q' over S', i.e. $pv_{max}(Q, S, \overline{D}) = pv_{max}(Q', S')$. The computation of $pv_{max}(Q, S, \overline{D})$ can still be performed in $O(|S| \cdot |Q|)$. The next lemma ensures that the subset-maximal arc-consistent pre-valuation on some domains $\overline{D}$ keeps all the answers in $D_1 \times \ldots \times D_n$. Let $ans_Q^S(\overline{D}) = (D_1 \times \ldots \times D_n) \cap Q(S)$ be the set of answers of Q on S using only values compatible with $\overline{D}$.

Lemma 5. *Let $\Theta = pv_{max}(Q, S, \overline{D})$. Then $ans_Q^S(\overline{D}) = ans_Q^S(\Theta(x_1) \times \ldots \times \Theta(x_n))$.*

Proof. As $pv_{max}(Q, S, \overline{D}) \subseteq D_1 \times \ldots \times D_n$, we have $ans_Q^S(pv_{max}(Q, S, \overline{D})) \subseteq ans_Q^S(\overline{D})$. Conversely, suppose that $(x_1, \ldots, x_n) \in ans_Q^S(\overline{D}) \setminus ans_Q^S(pv_{max}(Q, S, \overline{D}))$. Consider

$(D_1^{\Theta}, \ldots, D_n^{\Theta}) = pv_{max}(Q, S, \overline{D})$. Then the pre-valuation $(D_1^{\Theta} \cup \{x_1\}, \ldots, D_n^{\Theta} \cup \{x_n\})$ is arc-consistent, and bigger than $pv_{max}(Q, S, \overline{D})$, which contradicts its maximality. $\square$

We now present Algorithm 2, our enumeration algorithm. This algorithm outputs all elements of $Q(S)$ in lexicographical order w.r.t. $<$ (the order of $\underline{\mathrm{X}}$ relations) for the chosen order on the free variables of Q.

Algorithm 2 Enumeration algorithm for conjunctive queries over $\underline{\mathrm{X}}$ structures

```
    procedure MAIN($Q, S, <$)
2:      $(D, \mathcal{R}) \leftarrow S$;  $\tau \leftarrow$ FIRST($D, Q$)
        while $\tau \neq \bot$ do output($\tau$);  $\tau \leftarrow$ NEXT($\tau, D, Q$)
4:  function FIRST($D, Q$)
        $\Theta \leftarrow pv_{max}(Q, S, (D, \ldots, D))$
6:      if $\Theta \neq (\emptyset, \ldots, \emptyset)$ then return $\min_< \Theta$ else return $\bot$
    function NEXT($(v_1, \ldots, v_n), D, Q$)
8:      $j \leftarrow n - 1$
        repeat
10:         $\Theta \leftarrow pv_{max}(Q, S, (\{v_1\}, \ldots, \{v_j\}, D^{>}_{v_{j+1}}, D, \ldots, D))$;  $j \leftarrow j - 1$
        until $\Theta \neq (\emptyset, \ldots, \emptyset)$ or $j < 0$
12:     if $\Theta \neq (\emptyset, \ldots, \emptyset)$ then return $\min_< \Theta$ else return $\bot$
```

We first use pv_{max} on the whole domain D for all free variables, and get a first answer $(v_1, \ldots, v_n)$ by taking the minimum valuation. Then we exclude v_n from the domain of x_n, and all smaller elements, by running pv_{max} on the domains $(\{v_1\}, \ldots, \{v_{n-1}\}, D^{>}_{v_n})$, where $D^{>}_{v_i} = \{v \in D \mid v > v_i\}$. If no answer is returned, we run pv_{max} on $(\{v_1\}, \ldots, \{v_{n-2}\}, D^{>}_{v_{n-1}}, D)$, and so on. Proposition 3 shows that the solution returned by the function $next$ is indeed the next answer in $Q(S)$ in lexicographical order. This proves the correctness of Algorithm 2.

Proposition 3. *For all tuples of elements $(v_1, ..., v_n) \in D^n$, the successor of $(v_1, ..., v_n)$ by $<_{lex}$, if it exists, is* $\min\limits_{0 \leq j < n} \min\limits_{<} pv_{max}(Q, S, (\{v_1\}, \ldots, \{v_j\}, D^{>}_{v_{j+1}}, D, \ldots, D))$.

Proof. Let succ_{lex} denote the next solution in lexicographical order, that is to say: $\mathrm{succ}_{lex}(v_1, \ldots, v_n) = \min_< \{(v'_1, \ldots, v'_n) \in Q(S) \mid (v'_1, \ldots, v'_n) >_{lex} (v_1, \ldots, v_n)\}$. We have, for every $0 \leq j < n$,

$$\begin{aligned} &\min\nolimits_< ans(\{v_1\}, \ldots, \{v_j\}, D^{>}_{v_{j+1}}, D, \ldots, D) \\ &\quad = \min\nolimits_< ans(pv_{max}(Q, S, (\{v_1\}, \ldots, \{v_j\}, D^{>}_{v_{j+1}}, D, \ldots, D))) \\ &\quad = \min\nolimits_< pv_{max}(Q, S, (\{v_1\}, \ldots, \{v_j\}, D^{>}_{v_{j+1}}, D, \ldots, D)) \end{aligned}$$

The first equality is by Lemma 5, and the second by Lemma 4. Thus,

$$\begin{aligned} \mathrm{succ}_{lex}(v_1, \ldots, v_n) &= \min\nolimits_{0 \leq j < n} \min\nolimits_< ans(\{v_1\}, \ldots, \{v_j\}, D^{>}_{v_{j+1}}, D, \ldots, D) \\ &= \min\nolimits_{0 \leq j < n} \min\nolimits_< pv_{max}(Q, S, (\{v_1\}, ..., \{v_j\}, D^{>}_{v_{j+1}}, D, ..., D)) \end{aligned}$$

The first equality is due to the lexicographical order, and the second to the equalities above. $\square$

We call pv_{max} at most n times between two successive answers, *i.e.* Algorithm 2 has a delay in time $O(n \cdot |S| \cdot |Q|)$.

Theorem 1. *Let S be an $\underline{\text{X}}$ structure and Q an n-ary conjunctive query over S. Then $Q(S)$ can be enumerated without preprocessing, and with a delay in $O(n \cdot |S| \cdot |Q|)$ between two successive answers.*

4 Acyclic conjunctive queries over $\underline{\text{X}}$ structures

A conjunctive query Q is *acyclic* if it admits a join-tree [1], or equivalently (for binary relations) if the following undirected graph G_Q is acyclic: $G_Q = (V_Q, E_Q)$ with $V_Q = \text{vars}(Q)$ and E_Q is the set of edges s.t. $\{x, y\} \in E_Q$ iff $R(x, y)$ occurs in Q for some R. In this section, we present an enumeration algorithm for ACQS over $\underline{\text{X}}$ relations (ACQS($\underline{\text{X}}$)). It works with a preprocessing $O(|Q|\cdot|S|)$ and a delay $O(|Q|\cdot|D|)$, where Q is the query and D the domain. Then we show that when paying a preprocessing in time $O(|D|^2\cdot|Q|)$, one can reduce the delay to $O(n\cdot|D|)$, where n is the arity of the query. As the associated graph G_Q of an ACQ Q over a binary signature is nothing else than a forest, we define a notion of tree-like queries into which ACQS over a structure S can be naturally encoded (in linear-time), while preserving $\underline{\text{X}}$ properties of relations.

4.1 Tree patterns

Definition 1. *A tree pattern over a binary signature σ and a countable set of variables V is an ordered binary tree whose nodes are labeled in $V \cup \sigma \cup \sigma \times \sigma$. It is inductively defined by terms generated by the following grammar:*

$$T \ ::= \ x \mid R(T') \mid (R, R')(T_1, T_2) \qquad \textit{where } x \in V \textit{ and } R, R' \in \sigma$$

Moreover, the variables occurring at the leaves are all pairwise distinct.

The semantics of tree patterns over σ is given by means of ACQS over σ. Intuitively, every inner-node corresponds to an existentially quantified variable, every leaf to a free variable, and every branching to a conjunction. Therefore to define the semantics in terms of ACQS, one needs to introduce a new bound variable for every inner-node. We denote by $\text{vars}_{\text{free}}(T)$ the variables occurring in T (necessarily at the leaves). For any variable x and fresh variables $y, z, x' \notin \text{vars}_{\text{free}}(T)$, we denote by $Q_{T,x}$ the CQ:

$$Q_{T,x} \ = \ \begin{cases} x = y & \text{if } T = y \\ \exists x' \ R(x, x') \wedge Q_{T',x'} & \text{if } T = R(T') \\ \exists y \exists z \ R(x, y) \wedge R'(x, z) \wedge Q_{T_1,y} \wedge Q_{T_2,z} & \text{if } T = (R, R')(T_1, T_2) \end{cases}$$

The ACQ Q_T associated with a tree pattern T is defined by $Q_T = \exists x \cdot Q_{T,x}$, for any variable $x \notin \text{vars}_{\text{free}}(T)$ (the choice of the variable does not matter as equivalence is preserved when choosing another variable). E.g. let $T = (R_1, R_2)((R_3, R_4)(x_1, x_2), x_3)$. Then $Q_T(x_1, x_2, x_3) = \exists x \exists y R_1(x, y) \wedge R_3(y, x_1) \wedge R_4(y, x_2) \wedge R_2(x, x_3)$, for some variables x, y. Since the variables of T are all distinct, Q_T is acyclic. We extend the notion of answer set to tree patterns naturally. For a structure S over σ, $T(S) = Q_T(S)$. The size of a tree pattern is its number of nodes.

We now show that for any tree pattern T and any $\underline{\text{X}}$ σ-structure S with domain D, $T(S)$ can be enumerated with a preprocessing $O(|S| + |D|\cdot|T|)$ and a delay $O(|D|\cdot|T|)$.

Let $x \notin \text{vars}_{\text{free}}(T)$, and consider the ACQ $Q_{T,x}$ as defined before. For all $a \in D$, we denote by $Q_{T,a}$ the ACQ $Q_{T,x}$ where each occurrence of x is replaced by a. We denote by $T(S,a)$ the answer set $Q_{T,a}(S)$. Clearly, $T(S,a) \subseteq T(S)$. Informally, $T(S,a)$ is the set of tuples that can be obtained by mapping the root of T to a.

We denote by $\text{Sub}(T)$ the set of subtrees (subterms) of T. The first step of the algorithm is to compute a mapping $\texttt{sat} : \text{Sub}(T) \to 2^D$ such that for all $T' \in \text{Sub}(T)$ and for all $a \in D$, $a \in \texttt{sat}(T')$ iff $T'(S,a) \neq \emptyset$. Informally, $\texttt{sat}(T')$ is the set of elements such that there exists a solution of T' in S that can be obtained when mapping the root of T' to a. This mapping can be computed efficiently:

Lemma 6. *For every tree pattern T over σ and every $\underline{\text{X}}$ σ-structure S over a domain D given in sorted representation,* $\texttt{sat}$ *can be computed in time $O(|D|\cdot|T|)$.*

Proof. The mapping $\texttt{sat}$ can be computed inductively in a bottom-up manner, as

$$\begin{aligned}
\texttt{sat}((R_1,R_2)(T_1,T_2)) &= R_1^{-1}(\texttt{sat}(T_1)) \cap R_2^{-1}(\texttt{sat}(T_2))\\
\texttt{sat}(R_1(T_1)) &= R_1^{-1}(\texttt{sat}(T_1))\\
\texttt{sat}(x) &= D
\end{aligned}$$

The result follows by Lemma 3 and since intersection can be computed in $O(|D|)$. $\square$

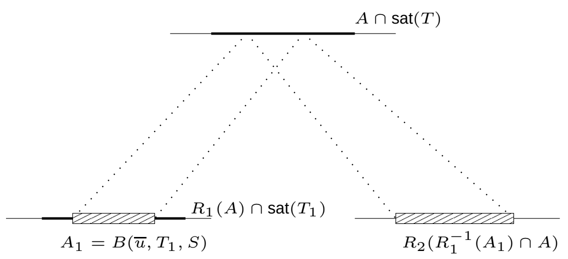
Similar techniques have also been used to evaluate XPath (unary) queries [12]. Let T be a tree pattern over a binary signature σ, and let $x_1,\dots,x_n$ be the variables occurring at the leaves of T in left-to-right order (*i.e.* from the left-most leaf to the right-most leaf). Given an $\underline{\text{X}}$ σ-structure S for some total order $<$ on the domain D, we define an algorithm that enumerates $T(S)$ in lexicographic order with respect to $x_1,\dots,x_n$ and $<$. We denote by $<_{lex}$ this order. For all $A \subseteq D$, we let $T(S,A) = \bigcup_{a\in A} T(S,a)$. Let $T' \in \text{Sub}(T)$ and $\overline{u} \in T'(S)$. The tuple $\overline{u}$ defines the set $B(\overline{u},T',S) = \{a \in D \mid \overline{u} \in T'(S,a)\}$. Informally, $B(\overline{u},T',S)$ is the set of nodes from which we can obtain $\overline{u}$.

Lemma 7. *If $T = (R_1,R_2)(T_1,T_2)$, then for all $A \subseteq D$:*

$$T(S,A) \;=\; \bigcup_{\overline{u}\in T_1(S,R_1(A))} \{\overline{u}\} \times T_2(S, R_2(R_1^{-1}(B(\overline{u},T_1,S)) \cap A))$$

Proof. Let $\overline{w} \in T(S,A)$. By definition of $T(S,A)$, there is $a \in A$ such that $\overline{w} \in T(S,a)$. Therefore there exists $a_1,a_2 \in D$ such that $\overline{w}$ can decomposed into $\overline{u}$ and $\overline{v}$, $(a,a_1) \in R_1$, $(a,a_2) \in R_2$, $\overline{u} \in T_1(S,a_1)$ and $\overline{v} \in T_2(S,a_2)$. Clearly, $a_1 \in R_1(A)$, and $a_1 \in B(\overline{u},T_1,S)$. Therefore $a \in R_1^{-1}(B(\overline{u},T_1,S)) \cap A$, and $a_2 \in R_2(R_1^{-1}(B(\overline{u},T_1,S)) \cap A)$. Therefore $\overline{v} \in T_2(S,R_2(R_1^{-1}(B(\overline{u},T_1,S)) \cap A)$.

Conversely, let us take two tuples $\overline{u}$ and $\overline{v}$ such that $\overline{u} \in T_1(S,R_1(A))$ and $\overline{v} \in T_2(S,R_2(R_1^{-1}(B(\overline{u},T_1,S))\cap A))$. Therefore there exists $a_2 \in R_2(R_1^{-1}(B(\overline{u},T_1,S))\cap A)$ such that $\overline{v} \in T_2(S,a_2)$. There is $a \in A \cap R_1^{-1}(B(\overline{u},T_1,S))$ such that $(a,a_2) \in R_2$. There is also $a_1 \in B(\overline{u},T_1,S)$ such that $(a,a_1) \in R_1$. Since $a_1 \in B(\overline{u},T_1,S)$, $\overline{u} \in T_1(S,a_1)$. Moreover, $a_1 \in R_1(A)$ since $(a,a_1) \in R_1$ and $a \in A$. Therefore we have found a,a_1,a_2 such that $a \in A$, $(a,a_1) \in R_1$, $(a,a_2) \in R_2$, $\overline{u} \in T_1(S,a_1)$ and $\overline{v} \in T_2(S,a_2)$. In other words, $\overline{u}.\overline{v} \in T(S,a) \subseteq T(S,A)$. $\square$



$A \cap \text{sat}(T)$
$R_1(A) \cap \text{sat}(T_1)$
$A_1 = B(\overline{u}, T_1, S)$
$R_2(R_1^{-1}(A_1) \cap A)$

Fig. 2. Branching management for tree patterns enumeration

Similar lemmas hold when T is a single variable node, or the root of T is branching-free. In particular, $T(S, A) = A$ if T is a variable. We now have the main ingredient of a recursive enumeration algorithm that we illustrate for the case $T = (R_1, R_2)(T_1, T_2)$: for each tuple $\overline{u} \in T_1(S, R_1(A))$ enumerated recursively in lexicographic order, we have to compute the set $A_1 = B(\overline{u}, T_1, S)$, and then the set $A_2 = R_2(R_1^{-1}(A_1) \cap A)$. Then we recursively enumerate the tuples $\overline{v}$ of $T_2(S, A_2)$ in lexicographic order. Instead of computing the set A_1 once $\overline{u}$ has been computed, A_1 can be computed recursively when applying the enumeration algorithm on T_1. This is because $B(\overline{w}, T, S) = R_1^{-1}(B(\overline{u}, T_1, S)) \cap R_2^{-1}(B(\overline{v}, T_2, S))$, where $\overline{w} = \overline{u}.\overline{v}$. The enumeration algorithm is therefore defined by a recursive procedure that outputs the next tuple $\overline{w}$ of $T(S, A)$ and outputs the set $B(\overline{w}, T, S)$. However it might be the case that A_2 is empty. In this case, one has to enumerate the tuples of $T_1(S, R_1(A))$ until there is a tuple $\overline{u}$ such that $R_2(R_1^{-1}(B(\overline{u}, T_1, S)) \cap A) \neq \emptyset$. This can lead to an unbounded delay between two consecutive tuples. Therefore we add one more constraint on the sets to ensure the following invariant: at each recursive call of the procedure, we must have $A \neq \emptyset$ and $A \subseteq \mathtt{sat}(T)$. Hence we are sure that there is at least one tuple in $T(S, A)$. If $A \subseteq \mathtt{sat}(T)$, when we call the procedure on T_1, instead of calling it on $T_1, S, R_1(A)$, we call it on $T_1, S, R_1(A) \cap \mathtt{sat}(T_1)$. Since $\emptyset \neq A$ and $A \subseteq \mathtt{sat}(T)$, $R_1(A) \cap \mathtt{sat}(T_1) \neq \emptyset$ and the invariant is satisfied. Similarly, for the right subtree, we call the procedure on $T_2, S, A_2 \cap \mathtt{sat}(T_2)$. This is depicted on Fig. 2.

If the root of T is branching-free, the enumeration works similarly. When T is reduced to a single node labeled by a variable x, the algorithm enumerates all elements a of A w.r.t. the order $<$ on D and for each element returns a and $\{a\}$ (*i.e.* $B(a, x, S)$).

The enumeration algorithm (Algorithm 3) is presented in a Python-like style, which allows us to write it in a very concise and readable way. In particular, we define an enumerator $\mathsf{ENUM}(T, A)$ that enumerates $T(S, A)$. The instruction *yield* passes its argument to the parent enumerator call, which outputs the yielded values and freezes the computation by storing the evaluation context. Therefore, when an instruction **for** $(\overline{u}, B) \in \mathsf{ENUM}(T, A)$ is executed, it passes through the loop each time $\mathsf{ENUM}(T, A)$ yields a new element. In other words, $\mathsf{ENUM}(T, A)$ is evaluated in a by-need lazy fashion. This comes without extra cost in time complexity.

Lemma 8 (Completeness and Soundness). *Given a tree pattern T and an $\underline{X}$ structure S for some total order $<$ on its domain D and a subset $A \subseteq D$, ENUM(T, A) enumerates all elements of $T(S, A)$ in lexicographic order, and only those tuples. Moreover for each enumerated tuple $\overline{u}$, it yields the set $B(\overline{u}, T, S)$.*

Algorithm 3 Enumeration algorithm for tree patterns over $\underline{X}$ structures

```
   function MAIN($T, S, <$)  ▷ $T$:tree pattern, $S$:$\underline{X}$ structure for some order $<$ on its domain $D$
2:     compute a sorted representation for $S$
       compute the function sat
4:     for $(\overline{u}, B) \in$ ENUM($T$, sat($T$)) do
           output $\overline{u}$
6: function ENUM($T, A$)
       if $T = (R_1, R_2)(T_1, T_2)$ then
8:         for $(\overline{u}_1, B_1) \in$ ENUM($T_1, R_1(A) \cap$ sat$(T_1)$) do
               for $(\overline{u}_2, B_2) \in$ ENUM($T_2, R_2(R_1^{-1}(B_1) \cap A) \cap$ sat$(T_2)$) do
10:                yield $(\overline{u}_1.\overline{u}_2,\ R_1^{-1}(B_1) \cap R_2^{-1}(B_2))$
       if $T = R_1(T_1)$ then
12:        for $(\overline{u}_1, B_1) \in$ ENUM($T_1, R_1(A) \cap$ sat$(T_1)$) do
               yield $(\overline{u}_1,\ R_1^{-1}(B_1))$
14:    if $T = x$ then
           for $a \in A$ w.r.t. $<$ do
16:            yield $(a,\ \{a\})$
```

Lemma 9. *Given a tree pattern T and an $\underline{X}$ structure S for some total order $<$ on its domain D and a set $A \subseteq D$ such that $A \neq \emptyset$ and $A \subseteq \mathtt{sat}(T)$, ENUM($T, A$) enumerates $T(S, A)$ with preprocessing in $O(|S| + |D|\cdot|T|)$ and delay in $O(|D|\cdot|T|)$.*

Proof. The first two steps (lines 2 and 3) are obtained by Lemma 2 and Lemma 6. This gives the preprocessing step.

For the delay, the proof is very similar for the cases of two consecutive tuples and first tuple, we do it for two consecutive tuples only. It is done by induction on T. If T is a leaf labeled x, then it is clear that all elements of A can be enumerated with a delay $O(|D|)$. Since $|T| = 1$, we get the result. If $T = (R_1, R_2)(T_1, T_2)$, by Lemma 8, we know that exactly all tuples of $T(S, A)$ are enumerated in lexicographic order. Let us take two consecutive tuples $\overline{u} <_{lex} \overline{v}$ such that $\overline{u}, \overline{v} \in T(S, A)$. Those two tuples can be decomposed into $\overline{u} = \overline{u_1}.\overline{u_2}$ and $\overline{v} = \overline{v_1}.\overline{v_2}$ where $\overline{u_1}$ is matched by T_1, $\overline{u_2}$ by T_2, $\overline{v_1}$ by T_1 and $\overline{v_2}$ by T_2. We consider two cases:

If $\overline{u_1} = \overline{v_1}$ and $\overline{u_2} <_{lex} \overline{v_2}$, then let $A_2 = R_2(R_1^{-1}(B_1) \cap A) \cap \mathtt{sat}(T_2)$ where B_1 is the set returned at line 8. We know by Lemma 8 that $B_1 = B(\overline{u_1}, T_1, S)$. We prove that $A_2 \neq \emptyset$ and $A_2 \subseteq \mathtt{sat}(T_2)$ (in order to apply the induction hypothesis). It is clear that $A_2 \subseteq \mathtt{sat}(T_2)$. By Lemma 8, $\overline{u_1} \in T(S, R_1(A))$, therefore there exists $a \in A$ and $a_1 \in R_1(A)$ such that $(a, a_1) \in R_1$ and $\overline{u_1} \in T(S, a_1)$. Moreover, $a_1 \in B(\overline{u_1}, T_1, S)$. Since $A \subseteq \mathtt{sat}(T)$, there exists a_2 such that $(a, a_2) \in R_2$ and $a_2 \in \mathtt{sat}(T_2)$. In particular, $a_2 \in A_2$ and $A_2 \neq \emptyset$. Moreover, $A_2 \subseteq \mathtt{sat}(T_2)$, therefore $T(S, A_2) \neq \emptyset$. Since by Lemma 8 the tuples are enumerated in lexicographic order, the tuple $\overline{v_2}$ is the successor of $\overline{u_2}$ in the set $T_2(S, A_2)$. Therefore by induction hypothesis, $\overline{v_2}$ is obtained after $\overline{u_2}$ with a delay $O(|D|\cdot|T_2|)$. *A fortiori*, $\overline{v_1}.\overline{v_2} = \overline{u_1}.\overline{v_2}$ is obtained with a delay $O(|D|\cdot|T_2|) = O(|D|\cdot|T|)$.

Suppose that $\overline{u_1} <_{lex} \overline{v_1}$. It is clear that $R_1(A) \cap \mathtt{sat}(T_1) \neq \emptyset$, since $A \neq \emptyset$ and $A \subseteq \mathtt{sat}(T_1)$. Therefore $T_1(S, R_1(A)) \neq \emptyset$. Since by Lemma 8 the tuples are enumerated in lexicographic order, $\overline{v_1}$ is necessarily the successor of $\overline{u_1}$ in the set $T_1(S, R_1(A))$. By induction hypothesis, it is obtained after a delay $O(|D|\cdot|T_1|)$. We let B_1 be the set returned at line 8 after $\overline{v_1}$ has been computed. By Lemma 8, we know that $B_1 = B(\overline{v_1}, T_1, S)$. Similarly as the previous case, one can show that the set $A_2 = R_2(R_1^{-1}(B_1) \cap A) \cap \mathtt{sat}(T_2)$ is non-empty and satisfies $A_2 \subseteq \mathtt{sat}(T_2)$. Therefore $T_2(S, A_2) \neq \emptyset$ and $\overline{v_2}$ is necessarily the first element of $T_2(S, A_2)$. By hypothesis, this first element can be obtained with a delay $O(|D|\cdot|T_2|)$. In order to give the overall delay to output $\overline{u_2}.\overline{v_2}$ after $\overline{u_1}.\overline{v_1}$, one finally needs to give the time complexity to compute the set A_2. Since we have first computed a sorted representation of S, by Lemma 3 all operations can be done in $O(|D|)$. The overall delay is therefore $O(|D|\cdot|T_1| + |D|\cdot|T_2| + |D|) = O(|D|\cdot|T|)$.

Finally, if the root of T is branching-free, the proof similar and easier than for binary branching. □

Therefore one obtains the following theorem:

Theorem 2. *For every tree pattern T and every $\underline{\mathrm{X}}$-structure S for some total order $<$ on its domain D, $T(S)$ can be enumerated with preprocessing $O(|S| + |T|\cdot|D|)$ and delay in $O(|T|\cdot|D|)$.*

As a matter of fact, the delay mentioned in the previous theorem can be reduced to $O(n\cdot|D|)$, where n is the number of free variables, with the cost of a preprocessing in $O(|T|\cdot|D|^2)$. This is done by transforming the tree pattern in a full binary tree: the branching-free paths are replaced by a unique edge. The source of this edge is then labeled by a relational predicate interpreted by the composition of all the relations occurring along the path. Therefore one changes the pattern and the structure on which its relational symbols are interpreted. As we have to perform the composition of $\underline{\mathrm{X}}$ relations, the time complexity of this reduction is $O(|T|\cdot|D|^2)$ (Prop. 1). The resulting tree pattern is a binary tree of size $O(n)$. Then we can apply Algorithm 3.

Theorem 3. *For every tree pattern T with n (free) variables and every $\underline{\mathrm{X}}$-structure S, $T(S)$ can be enumerated with a preprocessing $O(|T|\cdot|D|^2)$ and a delay in $O(n\cdot|D|)$.*

Remark 1. Note that Algorithm 3 also works for any kind of structure over binary predicates (if we remove the computation of a sorted representation). The complexity of the preprocessing and delay depends on the following operations: computing $R(A)$ and $R^{-1}(A)$ for any relation R and subset A of the domain. In the general case of ACQs over an arbitrary structure where the (binary) relations are represented as pairs of elements, $R(A)$ and $R^{-1}(A)$ can be computed in $O(|R| + |A|) = O(|S|)$. This results in an enumeration algorithm with preprocessing and delay $O(|S|\cdot|T|)$.

4.2 From ACQs to tree patterns

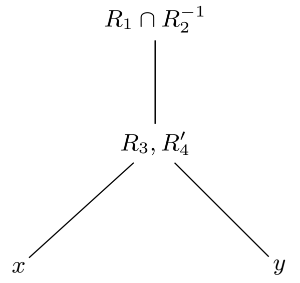
Given an acyclic conjunctive query Q and an $\underline{\mathrm{X}}$ structure S, one first transforms Q and S into a tree pattern T_Q and a structure S' with the same domain such that $|S'| = O(|Q|\cdot|S|)$ and $Q(S) = T_Q(S')$. Then we apply the enumeration algorithm for tree patterns. The transformation works on the labeled (directed) graph H_Q of Q defined by

$H_Q = (V_Q, E_Q, \lambda)$ where $V_Q = \text{vars}(Q)$, $E_Q = \{(x, y) \mid R(x, y) \in Q \text{ for some } R\}$ and for all $(x, y) \in E_Q$, $\lambda(x, y) = \{R \mid R(x, y) \in Q\}$. Since Q is acyclic, this graph is acyclic as well (acyclicity in this case being defined without considering the orientation of edges). Therefore it is a forest, but it is not a tree pattern for one (or more) of the following reasons: (i) there might be several disconnected components, (ii) edges are labeled by several relational symbols, (iii) a vertex may have several incoming edges, (iv) a free variable may not be a leaf, (v) the branching is arbitrary.

Suppose first that there is only one connected component. The first step is to choose a particular vertex that will be the root of the tree pattern. Then we have to adapt the orientation of the edges so that the unique path from the root to any vertex consists of edges that have the same orientation. This is done by taking the inverse of some relations (which remains $\underline{\text{X}}$) that are badly oriented. For instance, when changing the orientation of an edge (x, y) to (y, x), we change all its labels $R \in \lambda(x, y)$ by a new relation symbol R^{-1} that will later be interpreted by the inverse of R^S. The second step is to replace multiple labels by a single relational symbol that will denote the intersection of relations. For instance, if (x, y) is labeled by the following predicates $\{R_1, \ldots, R_k\} \subseteq \sigma$, we replace it by a new relational symbol $(R_1 \cap \cdots \cap R_k)$ that will later be interpreted as $\bigcap_i R_i^S$. Finally, free variables may not be necessarily at the leaves. Let x such that x is free but not a leaf. We replace x by some new variable x_1 in the query and add $\exists x_1 \cdot I(x_1, x)$ where I is a new relational symbol interpreted by the identity relational (which is $\underline{\text{X}}$ for any order). In the graph, it amounts to create a new vertex, to rename the vertex x by x_1 and to connect x_1 to x by an edge labeled I. By this transformation, all the free variables are at the leaves, but there are still leaves that are not free variables. We apply the following transformation exhaustively: if (x, y) is an edge labeled $R \in \sigma$ and every variable reachable from y is bound, then we remove the subtree t rooted at y and replace R by a new relational symbol $(R \cap t)$. It will be interpreted by $\{(u, v) \in R^S \mid v \in Q_t(S)\}$, where Q_t is the unary query represented by t, where y is considered as free (it can be evaluated in time $O(|D| \cdot |Q|)$ by using the same algorithm as in Lemma 6 and it is easy to see that the resulting relation is still $\underline{\text{X}}$). Applying this transformation exhaustively results in a tree whose leaves are all free variables. The resulting graph is almost a tree pattern, but its branching may be more than binary. Again we can duplicate some of its vertices to make it binary, by using the identity relational symbol I. The last step to get a tree pattern is to put the labels of the edges into their source node.

If there are several disconnected components, one first transforms each of them into a tree pattern, and create a new element connected to the roots of each tree pattern by a relation C interpreted for some $r \in D$ as $C^S = \{(r, d) \mid d \in D\}$ (it is $\underline{\text{X}}$). The branching is not binary but we can apply the same technique as before to get a binary tree.

The syntactical construction of the tree pattern can be done in time $O(|Q|)$. We have to compute a new structure S' in which every new introduced relational symbols is interpreted. This structure has the same domain as S. We assume that S is in sorted representation (done via a processing in $O(|S|)$). As shown by the construction, the interpretation of the new relation symbols is the result of taking intersection or inverse of $\underline{\text{X}}$ relations, as well as evaluating a unary acyclic conjunctive query over $\underline{\text{X}}$ relations,



$$\phi \equiv \exists x_1 \exists x_2 \exists x_3 .\ R_1(x_1, x_2) \wedge R_2(x_2, x_1) \wedge R_3(x_2, x) \wedge R_4(x_2, y) \wedge R_5(y, x_3)$$

R'_4 is interpreted as $\{(u, v) \mid R_4(u, v) \wedge \exists w.\ R_5(v, w)\}$.

Fig. 3. Tree pattern resulting from the translation of the ACQ ϕ.

which can again be done in $O(|D| \cdot |Q|)$. The new relational symbols are interpreted by relations of size $O(|S|)$ at most. Moreover, we have introduced at most $O(|Q|)$ new relational symbols. Therefore $|S'| = O(|S| \cdot |Q|)$. Finally, S' can be computed in time $O(|Q| \cdot |S|)$, as taking the intersection of two relations R_1, R_2 can be done in $O(|R_1| + |R_2|)$, taking the inverse is done in constant time (for sorted representation), and evaluating a unary query over S is in $O(|D| \cdot |Q|)$.

We provide an example in Fig. 3.

The complexity of this transformation depends on the complexity of intersection and inverse of relations, as well as evaluation of unary queries.

Lemma 10. *For every acyclic conjunctive query Q over an $\underline{\mathrm{X}}$ σ-structure S, one can construct in time $O(|S| \cdot |Q|)$ a tree pattern T_Q over a signature σ' and an $\underline{\mathrm{X}}$ σ'-structure S' with same domain such that $|T_Q| = O(|Q|)$, $|S'| = O(|S| \cdot |Q|)$ and $T_Q(S) = Q(S')$.*

As a corollary of Theorem 2, Theorem 3 and Lemma 10, we obtain

Theorem 4. *For every n-ary acyclic conjunctive query Q over an $\underline{\mathrm{X}}$ σ-structure S, $Q(S)$ can be enumerated with a preprocessing $O(|S| \cdot |Q|)$ and a delay $O(|Q| \cdot |D|)$. This delay reduces to $O(n \cdot |D|)$ with a preprocessing in $O(|D|^2 \cdot |Q|)$.*

Remark 2. The translation of ACQs to tree patterns also works for the general case of ACQs over an arbitrary structure of binary relations. Its complexity depends on the time needed to compute intersection and inverse of relations, as well as the time to evaluate unary queries. The latter is known to be in $O(|S| \cdot |Q|)$ [18], the former remains the same as the case of $\underline{\mathrm{X}}$. Therefore by Remark 1, we get an enumeration algorithm for general ACQs over a binary structure with a preprocessing $O(|S| \cdot |Q|)$ and a delay $O(|S| \cdot |Q|)$ (similar to that of [4]). Considering $\underline{\mathrm{X}}$ relations, this delay reduces to $O(|D| \cdot |Q|)$.

child	$= \mathit{fc} \circ \mathit{ns}^*$	parent	$= \text{child}^{-1}$
descendant	$= \text{child}^+$	ancestor	$= \text{descendant}^{-1}$
descendant-or-self	$= \text{descendant} \cup I_t$	ancestor-or-self	$= \text{descendant-or-self}^{-1}$
following-sibling	$= \mathit{ns}^+$	preceding-sibling	$= \text{following-sibling}^{-1}$
following	$= \text{ancestor-or-self} \circ \mathit{ns}^+ \circ$ descendant-or-self	preceding	$= \text{following}^{-1}$

Fig. 4. XPath axes

4.3 Enumeration of acyclic conjunctive XPath n-ary queries

In this section, we show that the ideas developed in the enumeration algorithm of ACQ ($\underline{\text{X}}$) can be adapted to an enumeration algorithm for ACQs over XPath axes interpreted on unranked trees. The case of XPath axes however differs in that the relations are not explicitly represented. *Unranked trees* is the widely accepted model of XML documents. In such trees, the nodes are labeled by elements of a finite alphabet Σ, sibling nodes are ordered, and a node may have an arbitrary number of children. We view unranked trees as a structure over the signature $\sigma_{unr} = \{(\mathit{lab}_a)_{a\in\Sigma}, \mathit{fc}, \mathit{ns}\}$ where for all $a \in \Sigma$, lab_a is a unary predicate that denotes the nodes labeled a, *fc* is a binary predicate that relates a node and its *first-child*, and *ns* is a binary predicate that relates a node and its *next-sibling*. For any unranked tree t, we let $\text{Dom}(t)$ be its set of nodes and $|t| = |\text{Dom}(t)|$ its number of nodes.

XPath axes are listed in Fig. 4 together with their semantics by means of expressions over inverse, union, composition and iteration $.^*$ and $.^+$ of the relations *fc*, *ns* and I_t the identity relation on $\text{Dom}(t)$. XPath axes are not $\underline{\text{X}}$, and only some subsets of them are $\underline{\text{X}}$, as shown in [13]. However as we will show, ACQs over XPath axes still enjoy good enumeration properties, mainly because of the following fact:

Lemma 11 (Gottlob, Koch, Pichler [12]). *For all unranked trees t, all XPath axes χ, and all sets $A \subseteq \text{Dom}(t)$, $\chi(A)$ can be computed in time $O(|t|)$.*

In the context of XPath queries, it is important to consider the unary predicates lab_a that test the labels of the nodes. We can slightly extend the tree patterns with optional unary predicates lab_a attached to the nodes of the tree pattern. They just restrict the domain of the variables (bound and free) of the associated ACQ. As the unary predicates can be integrated into the binary relations, Algorithm 3 can also be used for tree patterns with both unary and binary predicates.

Consider now a tree pattern T over the XPath axes and the unary predicates lab_a, $a \in \Sigma$, and an unranked tree t (represented by a σ_{unr}-structure). We can choose an arbitrary total order on the nodes and apply Algorithm 3 directly on t (without considering line 2). In contrast to ACQ ($\underline{\text{X}}$) however, the predicates that appear in T are not explicitly represented in the σ_{unr}-structure t (otherwise its size would be $O(|t|^2)$). Thanks to Lemma 11 and the fact that XPath axes are closed under inverse, tree patterns over XPath axes can be enumerated with a preprocessing and delay $O(|T| \cdot |t|)$.

When going from ACQs to tree patterns over XPath axes, we apply the same construction as for ACQ ($\underline{\text{X}}$). As XPath axes are closed under intersection and inverse, the resulting tree pattern is a tree pattern over XPath axes. Therefore we do not need to precompute the interpretation of the axes and we can apply the enumeration algorithm as done for tree patterns over XPath axes. We obtain the following complexity:

Theorem 5. *For every* ACQ Q *over the XPath axes and the unary predicates* $(lab_a)_{a\in\Sigma}$ *and all unranked tree* t *represented as a structure over ns and fc,* $Q(t)$ *can be enumerated with a preprocessing and delay in* $O(|Q|\cdot|t|)$.

5 Conjunctive queries with inequalities

In this section, one considers conjunctive queries over $\underline{\mathrm{X}}$ structures where in addition $\neq$ is allowed in the signature. Note that a conjunctive query with such inequalities is acyclic if the query obtained by ignoring inequalities is acyclic. In other words, inequalities play no role in defining acyclicity. We first show that even in the case of acyclic conjunctive queries, such queries are hard to evaluate for combined complexity. The proof is by reduction from POSITIVE 1-3 SAT [11].

Proposition 4. *The problem of checking whether a Boolean conjunctive query with inequalities is true on an* $\underline{\mathrm{X}}$ *structure is* NP*-complete for combined complexity. The result remains true even if the query restricted to the* $\underline{\mathrm{X}}$ *predicates is acyclic.*

Proof. Let us consider the following well-known NP-complete problem [11].

POSITIVE 1-3 SAT
Input: a positive 3-CNF formula φ
Question: is there a model of φ such that each clause is satisfied by exactly one variable?

Membership to NP is straightforward. We prove hardness by reduction from the problem POSITIVE 1-3 SAT. Let φ be a positive 3-CNF formula over variables $x_1, ..., x_n$. Let $c_1, ..., c_m$ be an enumeration of its clauses. For $i \leq n$, let $o(i)$ be the number of occurrences of x_i in φ.

First, one builds an ordering $\prec$ and an $\underline{\mathrm{X}}$ structure S for $\prec$ as follows. Structure S has for domain D the disjoint union of sets D_1,...,D_n, C_1,..., C_m that we now construct. For each variable x_i, subdomain D_i contains two elements x_i^0 and $\bar{x}_i^0$ with $x_i^0 \prec \bar{x}_i^0$. Let us consider an enumeration of clauses of φ and let $c_j = x_{j_1} \vee x_{j_2} \vee x_{j_3}$, $j \leq m$, be the jth clause. Suppose, that the rth, sth and tth occurrences of respectively x_{j_1}, x_{j_2} and x_{j_3} appears in c_j. One will denote the clause by $c_j = x_{j_1}^r \vee x_{j_2}^s \vee x_{j_3}^t$. One constructs a subdomain C_j containing the following elements in that order:

$$x_{j_1}^r \prec x_{j_2}^s \prec x_{j_3}^t \prec \alpha^j \prec \bar{x}_{j_1}^r \prec \bar{x}_{j_2}^s \prec \bar{x}_{j_3}^t \prec \beta^j \prec \gamma^j.$$

The ordering between sets is depicted by:

$$D_1 \prec \ldots \prec D_n \prec C_1 \prec \ldots C_m$$

It remains to describe relations on D. For each variable x_i, one introduces a relation $next_i$ which is made of two paths starting respectively from x_i^0 and $\bar{x_i}^0$ and joining the different occurrences of x_i and $\bar{x}_i$. More precisely :

– for all $0\leq k<o(i)$, $next_i(x_i^k, x_i^{k+1})$ and $next_i(\bar{x}_i^k, \bar{x}_i^{k+1})$

It is easy to check that the relations $next_i$ are $\underline{\mathrm{X}}$ for $\prec$. Finally, a relation C is introduced that maps for all $j \leq m$, $x^r_{j_1}$, $x^s_{j_2}$, $x^t_{j_3}$ to α^j and $\bar{x}^r_{j_1}$, $\bar{x}^s_{j_2}$, $\bar{x}^t_{j_3}$ to β^j and γ^j. Here again, this relation is $\underline{\mathrm{X}}$ for $\prec$.

We now construct the following query Q:

$$\begin{array}{l} (\exists x^j_i)^{j=0,\dots,o(i)}_{i=1,\dots,n} (\exists a^j b^j c^j)^{j=1,\dots,m} \\ \bigwedge^n_{i=1} D_i(x^0_i) \wedge \bigwedge^n_{i=1} \bigwedge_{0 \leq j < o(i)} next_i(x^j_i, x^{j+1}_i) \wedge \\ \bigwedge_{c_j = x^r_{j_1} \vee x^s_{j_2} \vee x^t_{j_3}} C(x^r_{j_1}, a^j) \wedge C(x^s_{j_2}, b^j) \wedge C(x^t_{j_3}, c^j) \wedge \\ a^j \neq b^j \wedge a^j \neq c^j \wedge b^j \neq c^j \end{array}$$

The formula states that there exists an assignment of variables (those chosen as x^j_i are set to true, the variables y^j_i are set to false) such that, for each i, following the path to each clause ends in three distinct elements a^j, b^j and c^j whose only interpretation can be α^j, β^j and γ^j. This means that only one over the three paths corresponds to a positive variable. Note that the structure and the formula have comparable sizes. Note also, that once restricted to $\underline{\mathrm{X}}$ predicates, the constructed formula is acyclic. $\square$

In contrast with the preceding result, we show that the hardness only relies roughly on the number of variables involved in at least one inequality.

Theorem 6. *Let S be an $\underline{\mathrm{X}}$ structure for some order $<$, let Q be an n-ary conjunctive (resp. acyclic conjunctive) query with inequalities with at most ℓ variables involved in at least one inequality. Then, $Q(S)$ can be enumerated with a delay $O(\ell^{O(\ell)} \cdot |Q| \cdot n \cdot |S| \cdot \log |D|)$ (resp. a delay $O(\ell^{O(\ell)} \cdot |Q| \cdot |D| \cdot \log |D|)$ and preprocessing cost in $O(|Q| \cdot |S|)$).*

Proof. The bound is obtained by partial application of techniques related to the color coding method of [2]. We will construct $h = O(\ell^\ell \cdot \log |D|)$ conjunctive (resp. acyclic conjunctive) queries Q_i, $i = 1, \dots, h$, on some $\underline{\mathrm{X}}$-structures S_i for order $<$ such that $Q(S) = \bigcup_{i \leq h} Q_i(S_i)$. Is it known (see for example [4]) that if each predicate of a union of size h can be enumerated by a bounded delay algorithm for some delay k and w.r.t. the same order, here $<_{lex}$, then the union can be enumerated by a bounded delay algorithm with delay $O(h \cdot k)$ for this same order. Hence the result will follow.

More precisely, the body of Q can be written as $Q^0 \wedge \bigwedge_{(i,j) \in I} x_i \neq x_j$ for some set of pairs I, where Q^0 is acyclic (if Q is acyclic) and free of inequalities. We write $\{x_1, \dots, x_\ell\}$ for the variables appearing in inequalities (some of them may be free in Q), and $[\ell]$ for $\{1, \dots, \ell\}$.

Let $\lambda : D \longrightarrow [\ell]$, be a proper ℓ coloring of D. Let (S, λ) be the extension of S by the coloring λ with each color i encoded by a new monadic predicate U_i. Obviously, if two elements have two different colors in a proper coloring then they are distinct. Let us consider query Q' whose body is: $Q^0 \wedge \bigwedge_{(i,j) \in I} \bigwedge^\ell_{k=1} \neg(U_k(x_i) \wedge U_k(x_j))$

Claim. One can enumerate the elements of $Q'(S, \lambda)$ with delay $O(\ell^\ell \cdot |Q| \cdot |S|)$. Moreover, if Q is acyclic then the delay can be improved to $O(\ell^\ell \cdot |Q| \cdot |D|)$.

Proof (of the claim). Since the interpretation is taken on a structure where the coloring is proper, then the number of possible colorings for $x_1, ..., x_\ell$ compatible with $\bigwedge_{(i,j) \in I} \bigwedge^\ell_{k=1} \neg(U_k(x_i) \wedge U_k(x_j))$ is bounded by ℓ^ℓ. The query Q' is equivalent to a disjunction of conjunctive queries Q_f of body $Q^0 \wedge U_{f(1)}(x_1) \wedge \cdots \wedge U_{f(\ell)}(x_\ell)$ for all functions $f : [\ell] \to [\ell]$ such that $f(i) \neq f(j)$ for all $(i,j) \in I$. Each Q_f is acyclic if Q^0 is acyclic. The result follows from Theorem 1 and 4. $\square$

It is known (see [2] and also [10]) that there exists an ℓ-perfect family Λ of size $2^{O(\ell)} \cdot \log |D|$ of hash functions from D to $[\ell]$, *i.e.* Λ is such that for every $C \subseteq D$ with $|C| = \ell$, there exists $\lambda \in \Lambda$ such that $\lambda(c) \neq \lambda(c')$ for all distinct $c, c' \in C$ (*i.e.* the restriction of λ to C is one-to-one). The following holds: $Q(S) = \bigcup_{\lambda \in \Lambda} Q'(S, \lambda)$. Clearly, if a tuple $\overline{a} = (a_1, \ldots, a_n) \in Q'(S, \lambda)$ for some $\lambda \in \Lambda$ then $\overline{a} \in Q(S)$. Conversely, let $\overline{a} \in Q(S)$ and A be a satisfying assignment of variables of Q such that the free variables of Q are assigned to $\overline{a}$. Let b_i be the assignment of x_i, $i = 1, ..., \ell$ in A. Then, it holds that $\bigwedge_{(i,j) \in I} b_i \neq b_j$. As λ is an ℓ-perfect family, there exists $\lambda \in \Lambda$ such that all distinct elements among $b_1, ..., b_\ell$ have distinct images (*i.e.* colors) by λ. Then $\overline{a} \in Q(S, \lambda)$. Then, the theorem follows by enumerating the union $\bigcup_{\lambda \in \Lambda} Q'(S, \lambda)$. $\square$

Conclusion. As a conclusion, we would like to address some further questions. First, we would like to characterise the complexity of the enumeration algorithms in terms of amortized delay, which we conjecture is smaller than the worst-case delay. Another question is to see whether the delays are tight. Finally, we will investigate the generalization to relations of arbitrary arity, as the $\underline{\text{X}}$ notion can be extended to n-ary relations.

Acknowledgments. We thank Joachim Niehren for fruitful discussions. This work was partially supported by the project ANR ENUM (ANR-07-BLAN-0327).

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. N. Alon, R. Yuster, and U Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.
3. G. Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Computer Science Logic*, LNCS 4646, pp 208–222. Springer, 2006.
4. G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *CSL*, LNCS 4646, pp 208–222. Springer, 2007.
5. N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *Proceedings of the ACM SIGMOD*, pp 310–321, 2002.
6. F. Bry, T. Furche, B. Linse, and A. Schröder. Efficient evaluation of n-ary conjunctive queries over trees and graphs. In *Workshop on Web Information and Data Mining*, 2006.
7. B. Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 2007.
8. T. Feder, P. Hell, J. Huang, and A. Rafiey. Adjusted interval digraphs. *Electronic Notes in Discrete Mathematics*, 32:83 – 91, 2009.
9. E. Filiot, J. Niehren, J.-M. Talbot, and S. Tison. Polynomial time fragments of XPath with variables. In *ACM Symposium on Principles of Database Systems*, pp 205–214. 2007.
10. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006.
11. M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completness*. W.H. Freeman and Co, San Francisco, 1979.
12. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005.

13. G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. *Journal of the ACM*, 53(2):238–272, 2006.
14. W. Gutjahr, E. Welzl, and G. Woeginger. Polynomial graph-colorings. *Discrete Applied Mathematics*, 35:29–45, 1992.
15. P. Hell and J. Nešetřil. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2(3):143 – 163, 2008.
16. Christoph Koch. Processing queries on tree-structured data efficiently. In *ACM Symposium on Principles of Database Systems*, pp 213–224, 2006.
17. J. Opatrny. Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114, 1979.
18. M. Yannakakis. Algorithms for acyclic database schemes. In *Proceeding of VLDB*, pp 82–94. IEEE Computer Society, 1981.