Between Tree Patterns and Conjunctive Queries: Is There Tractability beyond Acyclicity?

Filip Murlak, Michał Ogiński, and Marcin Przybyłko

Institute of Informatics, University of Warsaw
fmurlak@mimuw.edu.pl, {M.Oginski,M.Przybylko}@students.mimuw.edu.pl

Abstract. In static analysis of queries over trees in the presence of schemas, there is an exponential complexity gap between conjunctive queries (CQs, positive existential first-order formulae without disjunction) and tree patterns (tree-like acyclic CQs). Motivated by applications in XML data management, we consider various restrictions of CQs that bring their complexity down to that of tree patterns. Most importantly, we show that vertical tree patterns can be costlessly extended with full horizontal CQs over children. We also consider restricted classes of schemas and show that under disjunction-free schemas the complexity of static analysis sometimes drops dramatically.

1 Introduction

Static analysis is a common name used in database theory for problems that do not deal with data, but only with queries. Such problems are often part of complex data management tasks, like data integration and data exchange [15, 19]. Most important static analysis problems include satisfiabiliy (Given a query q, is there a database D such that the returned set of tuples $q(D)$ is nonempty?) and query containment (Given q_1, q_2, does $q_1(D) \subseteq q_2(D)$ hold for each D?). As for first order logic these problems are undecidable, restricted query languages are considered. For relational databases, conjunctive queries (CQs, positive existential formulae without disjunction) and their unions (UCQs) are used most widely. The reason is a relatively low cost of static analysis [12], and expressive power meeting most typical needs (select-from-where SQL queries). In contrast, in XML static analysis, where problems are relativized to XML trees accepted by a given schema (often modelled as a tree automaton), the complexity of full CQs, using child and descendant relations, and sibling order, is prohibitively high [8, 9, 16]. As a remedy, more restrictive languages of acyclic CQs and tree patterns (tree-like acyclic CQs) were introduced. For instance, literature on XML data exchange and metadata management considers almost exclusively tree patterns [1–3, 14]. A fine complexity analysis for CQs and UCQs over XML trees would be useful in designing richer formalisms, based on intermediate classes of queries.

Most research on static analysis for queries over XML trees was done for fragments of XPath 1.0, which is a language allowing only acyclic queries [5, 18, 20, 21, 24, 25], or XPath 2.0, which allows path intersection, but not

B. Rovan, V. Sassone, and P. Widmayer (Eds.): MFCS 2012, LNCS 7464, pp. 705–717, 2012.

arbitrary joins [11, 17]. As has been observed by Gottlob, Koch, and Shulz, each CQ on trees can be translated to a union of exponentially many polynomial tree patterns [16]. This gives an upper bound on the complexity of containment exponentially higher then that for tree patterns: 2ExpTime in general and ExpSpace under non-recursive schemas (where the depth of trees is bounded by the size of the schema), while for tree patterns they are ExpTime and PSpace, respectively [5, 20, 24]. Björklund, Martens, and Schwentick show that the exponential gap cannot be avoided in general, as even containment of CQs using only child and descendant relation is 2ExpTime-complete, and ask if there are more manageable classes of CQs, other then acyclic CQs [9]. We are most interested in results of the form: under certain restrictions, the complexity of containment of UCQs is the same as that of unions of tree patterns. For example, if only child relation is available, the general case reduces to the acyclic case, as each CQ can be rewritten as a single tree pattern of linear size (cf. [4]).

We focus on the restrictions most commonly studied in XML data exchange and metadata management: non-recursive or disjunction-free schemas (cf. nested-relational DTDs [2, 3, 6]), and limited use of horizontal or vertical relations. We first prove that, over words, containment of UCQs is PSpace-complete, just like for unions of tree patterns (Sect. 4). Then we apply these results to trees (Sect. 5) and show that the complexities match for a fairly general class of "forest-like" UCQs, combining vertical tree patterns with arbitrary horizontal CQs over children. This is further exploited to prove the same for

- UCQs that do not use the descendant relation;
- UCQs specifying labels of all mentioned nodes, under non-recursive DTDs.

Finally in Sect. 6 we show that under disjunction-free schemas the containment of UCQs without the next-sibling relation is in coNExpTime and PSpace-hard, and if additionally schemas are non-recursive, it is on the second level of the polynomial hierarchy; with next-sibling, the complexity does not drop.

We work exclusively with Boolean queries; as explained in [9], this is not a restriction. Due to space limitations some arguments are omitted. For more details see the appendix available at `www.mimuw.edu.pl/~fmurlak/papers/patsat.pdf`.

2 Preliminaries

XML documents and trees. We model XML documents as unranked labelled trees. Formally, a *tree* over a finite labelling alphabet Γ is a relational structure $\mathcal{T} = \langle T, \downarrow, \downarrow^+, \rightarrow, \overset{+}{\rightarrow}, (a^{\mathcal{T}})_{a\in\Gamma}\rangle$, where

- the set T is an unranked tree domain, i.e., a prefix-closed subset of $\mathbb{N}^*$ such that $n \cdot i \in T$ implies $n \cdot j \in T$ for all $j < i$;
- the binary relations $\downarrow$ and $\rightarrow$ are the child relation ($n \downarrow n \cdot i$) and the next-sibling relation ($n \cdot i \rightarrow n \cdot (i+1)$);
- $\downarrow^+$ and $\overset{+}{\rightarrow}$ are transitive closures of $\downarrow$ and $\rightarrow$;
- $(a^{\mathcal{T}})_{a\in\Gamma}$ is a partition of the domain T into possibly empty sets.

We write $|\mathcal{T}|$ to denote the number of nodes of tree $\mathcal{T}$. The partition $(a^{\mathcal{T}})_{a\in\Gamma}$ defines a labelling of the nodes of $\mathcal{T}$ with elements of Γ, denoted by $\ell_{\mathcal{T}}$.

Automata and DTDs. The principal schema language we use are tree automata, abstracting Relax NG [22, 23]. We are using a variant in which the state in a node v depends on the states in the previous sibling and the last child of v. Such automata are equivalent to standard automata on unranked trees, as explained in [23]. Formally, an automaton is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$, where Σ is the labelling alphabet (the set of element types in our case), Q is the state space with the initial state q_0 and final states F, and $\delta \subseteq Q \times Q \times \Sigma \times Q$ is the transition relation. A *run* of $\mathcal{A}$ over a tree $\mathcal{T}$ is a labelling ρ of the nodes of $\mathcal{T}$ with the states of $\mathcal{A}$ such that for each node v with children $v{\cdot}0, v{\cdot}1, \ldots, v{\cdot}k$ and previous sibling w, $\big(\rho(w), \rho(v \cdot k), \ell_{\mathcal{T}}(v), \rho(v)\big) \in \delta$. If v has no previous sibling, $\rho(w)$ in the condition above is replaced with q_0. Similarly, if v has no children, $\rho(v \cdot k)$ is replaced with q_0. The language of trees *recognized* by $\mathcal{A}$, denoted by $L(\mathcal{A})$, consists of all trees admitting an *accepting* run of $\mathcal{A}$, i.e. a run that assigns one of the final states to the root.

A simpler schema language is provided by DTDs. A *document type definition* (DTD) over a labelling alphabet Γ is a pair $D = \langle r, P_D \rangle$, where $r \in \Gamma$ is a distinguished root symbol and P_D is a function assigning regular expressions over $\Gamma - \{r\}$ to the elements of Γ, usually written as $\sigma \to e$, if $P_D(\sigma) = e$. A tree $\mathcal{T}$ *conforms to* a DTD D, denoted $\mathcal{T} \models D$, if its root is labelled with r and for each node s in $\mathcal{T}$ the sequence of labels of its children is in the language of $P_D(\ell_{\mathcal{T}}(s))$. The set of trees conforming to D is denoted by $L(D)$. It is well known (and easy to see) that there is a PTIME translation from DTDs to automata.

We shall often consider *non-recursive* schemas, DTDs or automata. A DTD D is non-recursive if in every tree conforming to D each path contains each label at most once. A schema given by a tree automaton is non-recursive if in each run (accepting or not) each path contains each state at most once. The height of trees conforming to non-recursive schemas is bounded by the size of the schema.

CQs and Patterns. A conjunctive query (CQ) over alphabet Γ is a formula of first order logic using only conjunction and existential quantification, over unary predicates $a(x)$ for $a \in \Gamma$ and binary predicates $\downarrow, \downarrow^+, \to, \overset{+}{\to}$ (referred to as *child*, *descendant*, *next sibling*, and *following sibling*, respectively). Since we work only with Boolean queries, to avoid unnecessary clutter we often skip the quantifiers, assuming that all variables are by default quantified existentially.

An alternative way of looking at CQs is via patterns. A *pattern* π over Γ can be presented as $\pi = \langle V, E_c, E_d, E_n, E_f, \ell_\pi \rangle$ where ℓ_π is a partial function from V to Γ, and $\langle V, E_c \cup E_d \cup E_n \cup E_f \rangle$ is a finite graph whose edges are split into child edges E_c, descendant edges E_d, next-sibling edges E_n, and following-sibling edges E_f. By $|\pi|$ we mean the size of the underlying graph.

We say that a tree $\mathcal{T} = \langle T, \downarrow, \downarrow^+, \to, \overset{+}{\to}, (a^{\mathcal{T}})_{a \in \Gamma} \rangle$ *satisfies* a pattern $\pi = \langle V, E_c, E_d, E_n, E_f, \ell_\pi \rangle$, denoted $\mathcal{T} \models \pi$, if there exists a homomorphism $h \colon \pi \to \mathcal{T}$, i.e., a function $h : V \to T$ such that

- $h : \langle V, E_c, E_d, E_n, E_f \rangle \to \langle T, \downarrow, \downarrow^+, \to, \overset{+}{\to} \rangle$ is a homomorphism of relational structures; and
- $\ell_{\mathcal{T}}(h(v)) = \ell_\pi(v)$ for all v in the domain of ℓ_π.

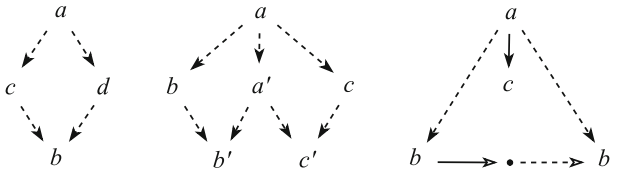

Fig. 1. Typical patterns. Void and solid heads indicate horizontal and vertical order, respectively. Dashed lines indicate transitive closure. The leftmost pattern can be expressed using path intersection operator from XPath 2.0, but the middle one cannot.

Each pattern can be seen as a CQ, and vice versa. In what follows we use the terms "pattern" and "CQ" interchangeably. *Tree patterns* are patterns whose underlying graph is a directed tree with edges pointing from parents to children.

Containment and Satisfiability. We focus on the following satisfiability problem.

PROBLEM: BC-SAT
INPUT: Boolean combination of patterns φ, schema $\mathcal{S}$ (automaton or DTD).
QUESTION: Is there a tree $\mathcal{T} \in L(\mathcal{S})$ such that $\mathcal{T} \models \varphi$?

For $\sigma \subseteq \{\downarrow, \downarrow^+, \rightarrow, \overset{+}{\rightarrow}, \_\}$ we write BC-SAT(σ) to denote BC-SAT restricted to patterns which use only the axes listed in σ. If the wildcard symbol, $\_$, is not present in σ, the labelling functions in patterns are required to be total, or in other words that each variable must occur in an atom of the form $a(x)$ for some $a \in \Gamma$. We use the following abbreviations: $\Downarrow$ for $\downarrow, \downarrow^+, \_$; and $\Rightarrow$ for $\rightarrow, \overset{+}{\rightarrow}, \_$.

Containment for UCQs is inter-reducible with non-satisfiability for Boolean combinations of the form $\pi \wedge \neg\pi_1 \wedge \neg\pi_2 \wedge \cdots \wedge \neg\pi_k$. Validity of a query is equivalent to non-satisfiability of its negation. Most of our lower bounds only use conjunctions of negations of patterns, thus giving dual lower bounds for validity (and containment).

3 Basic Complexity Bounds

In this section we briefly summarize known complexity bounds and establish some new bounds in the case of non-recursive schemas. The complexity of BC-SAT for tree patterns follows immediately from the results on XPath satisfiability and containment.

Theorem 1 ([5, 11, 20, 24]). *For tree patterns,* BC-SAT($\Downarrow, \Rightarrow$) *is* EXPTIME*-complete and* PSPACE*-complete under non-recursive schemas. The lower bounds hold even for containment of unions of tree patterns using only* $\downarrow$*.*

The EXPTIME upper bound follows from the translation of downward XPath to automata [24], later extended to cover horizontal axes in [11] (also implicit

in [20]). The PSPACE upper bound follows from [5]. The lower bounds in [5, 24] rely on the availability of wildcard (or disjunction inside XPath expressions), but they can be strengthened to queries using only $\downarrow$.

As we have mentioned, for CQs the bounds are exponentially worse.

Theorem 2 ([9]). *For arbitrary patterns,* $\mathsf{BC\text{-}SAT}(\Downarrow, \Rightarrow)$ *is* 2EXPTIME*-complete. The lower bound holds already for validity of a single CQ using only* $\Downarrow$*.*

To complete the background for the main results of this paper, described in Sections 4–6, we now settle the complexity of BC-SAT under non-recursive schemas. The complexity drops only slightly, but the cost of $\rightarrow$ begins to show.

Theorem 3. *Under non-recursive schemas* $\mathsf{BC\text{-}SAT}(\Downarrow)$ *and* $\mathsf{BC\text{-}SAT}(\Downarrow, \overset{+}{\rightarrow})$ *is* NEXPTIME*-complete and* $\mathsf{BC\text{-}SAT}(\Downarrow, \Rightarrow)$ *is* EXPSPACE*-complete. The lower bounds hold already for conjunctions of negated patterns.*

The EXPSPACE upper bound follows immediately by translation to tree patterns. The NEXPTIME upper bound is obtained via a linearly-branching model property, which relies on the fact that an unsatisfied pattern without $\rightarrow$ never becomes satisfied when a subtree is deleted. The lower bounds use an ingenious pattern construction from [9].

4 CQs over Words

In the classification sketched out in the previous section, horizontal CQs were to some extent drowned in the overall complexity of patterns. We shall have a closer look at them now: we restrict our models to words. We show that under this restriction the complexity of CQs matches that of tree patterns (Theorem 4); in the next section we show how this can be applied to the tree case.

Our main building block is a procedure $\textsc{Match}_\pi$ associated with each pattern π. The procedure takes as input a word w and checks if $w \models \pi$. It reads w letter by letter, possibly storing some information in the working memory, polynomial in $|\pi|$ and independent of w (it can be seen as a DFA, exponential in π). The procedure looks for the *earliest (leftmost) matching* of π in w, as defined below. We note that earliest matchings were previously used in [7] for tree patterns.

We use $\leq$ and $+1$ for the standard order and successor on the positions of words (an initial segment of natural numbers), and define homomorphisms just like for trees. Whenever we write $h : \pi \rightarrow w$, we implicitly assume that h is a homomorphism.

Definition 1. *Let* $g, h \colon \pi \rightarrow w$ *be two homomorphisms.*

- *We write* $g \leq h$ *if* $g(v) \leq h(v)$ *for each vertex* v *of* π*.*
- *We define* $\min(g, h) \colon \pi \rightarrow w$ *as* $\min(g, h)(v) = \min(g(v), h(v))$*.*

Lemma 1. *Let* w *be a word satisfying a* $\Rightarrow$*-pattern* π*.*

1. *For all* $g, h\colon \pi \to w$, $\min(g,h)$ *is a homomorphism.*
2. *There exists* $h_{\min}\colon \pi \to w$ *such that* $h_{\min} \le h$ *for all* $h\colon \pi \to w$.
3. *For each set* X *of vertices of* π *and each* $h\colon \pi \to w$ *there is a* $\hat{h}\colon \pi \to w$ *extending* $h|_X$ *such that* $\hat{h} \le h'$ *for each* $h'\colon \pi \to w$ *extending* $h|_X$.

We call the unique $h_{\min}$ from Lemma 1 the *earliest matching* of π in w.

$\textsc{Match}_\pi(w)$ works with components of π, called *firm subpatterns*, described in Definition 3.

Definition 2. *A* $\to$-component *of* π *is a maximal connected subgraph of* $\to$-*graph of* π. *In the* graph of $\to$-components of π, *denoted* G_π, *there is an edge from a* $\to$-*component* π_1 *to a* $\to$-*component* π_2 *if there is a* $\overset{+}{\to}$ *edge in* π *from a vertex of* π_1 *to a vertex of* π_2.

Definition 3. *A pattern* π *is* firm *if* G_π *is strongly connected. In general, each strongly connected component* X *of* G_π *defines a* firm subpattern *of* π: *the subgraph of* π *induced by the vertices of* $\to$-*components contained in* X. *The* DAG of firm subpatterns *of* π, *denoted* F_π, *is the standard DAG of strongly connected components of* G_π.

For example, the pattern in Fig. 2 on page 712 is firm, but has three $\to$-components.

The matching procedure $\textsc{Match}_\pi(w)$ works as follows:

- it reads the input word w from left to right trying to match firm subpatterns of π in the topological order given by F_π;
- for each firm subpattern it finds the earliest matching that does not violate the $\overset{+}{\to}$ edges connecting it with previously matched firm subpatterns.

Since we are proceeding in the topological order, each firm subpattern is processed after all its predecessors have been matched. Hence, the algorithm always finds a correct homomorphism or none at all. Completeness of the algorithm follows from the lemma below by straightforward induction (where Y is the union of previously matched firm patterns and X is obtained by adding a new one).

Lemma 2. *Let* $h\colon \pi \to w$ *be the earliest matching and let* X *be a set of vertices of* π *such that no edge enters* X *from the outside, and the only edges leaving* X *are* $\overset{+}{\to}$. *For each* $Y \subseteq X$, *if* $g\colon \pi|_X \to w$ *is the least homomorphism extending* $h|_Y$ *to* X, *then* $h|_X = g$.

Now we need to bound the memory used by $\textsc{Match}_\pi(w)$. We claim that the algorithm only needs to remember last $|\pi|$ symbols read (plus the matching constructed so far, restricted to this suffix). It is straightforward to check that each homomorphic image of a firm pattern π_0 is a subword of length at most $|\pi_0|$. Based on this observation, we prove the claim. For $i \le |w|$, let Π_i be the set of firm subpatterns of π matched by $\textsc{Match}_\pi(w)$ after processing the first i symbols of w. Note that if a position j is not touched by the matching, all firm subpatterns matched before this position are in Π_j. By pigeon-hole principle,

there is a position j between $i - |\pi|$ and i that is not touched by the matching. By the previous comment, all patterns from $\Pi_i \setminus \Pi_{i-1}$ are matched between j and i. It follows that $\textsc{Match}_\pi(w)$ only needs to remember the last $|\pi|$ symbols.

Using the matching procedure we prove the main result of this section.

Theorem 4. *On words* $\mathsf{BC\text{-}SAT}(\Rightarrow)$ *is* PSPACE*-complete, with hardness already for conjunctions of negated tree patterns using only* $\to, \_$ *or CQs using only* $\to, \overset{+}{\to}$.

Proof. To check if a Boolean combination φ is satisfiable in a word accepted by an automaton $\mathcal{A}$, we non-deterministically generate letters of a word $w \in L(\mathcal{A})$ and feed with them $\textsc{Match}_\pi$ for each π used in φ. We accept if the split into matched and unmatched patterns satisfies φ. To prevent looping, we count the number of letters and stop when we reach certain threshold, single exponential in $|\varphi|$. To establish the threshold, recall that $\textsc{Match}_\pi$ can be seen as a DFA, exponential in $|\pi|$. The product automaton corresponding to all running copies of the matching procedure is single exponential in φ. The threshold can be set to the size of the product automaton. By Savitch theorem we can eliminate non-determinism from this algorithm.

For the lower bound we give a reduction from the following tiling problem, which is known to be PSPACE-complete: Given a set of tiles $T = \{t_1, t_2, \ldots, t_k\}$, relations $H, V \subseteq T \times T$, and a number n in unary, decide if there is a number m and an $m \times n$ matrix $(a_{i,j})$ with entries from T such that $a_{1,1} = t_1$, $a_{m,n} = t_k$, $(a_{i,j}, a_{i,j+1}) \in H$ for $1 \leq i \leq m$, $1 \leq i < n$ and $(a_{i,j}, a_{i+1,j}) \in V$ for $1 \leq i < m$, $1 \leq i \leq n$. In fact we give the reduction from the following linearised tiling to which the original problem can be easily reduced: Given T, H, V, n, decide if there is a sequence of tiles $s_1 s_2 \ldots s_\ell$ such that $s_1 = t_1$, $s_\ell = t_k$, $(s_i, s_{i+1}) \in H$ for all $i \leq \ell - 1$, and $(s_i, s_{i+n}) \in V$ for all $i \leq \ell - n$.

Let an instance of the linearised tiling problem be T, H, V, n. If wildcard is available, we can assume our alphabet is $T \cup \{r\}$ and take the DTD $r \to t_0 T^* t_k$ and the following combination of patterns:

$$\bigwedge_{(t_i,t_j)\notin H} \neg\exists x\, \exists y\ (x \to y) \wedge t_i(x) \wedge t_j(y) \wedge \bigwedge_{(t_i,t_j)\notin V} \neg\exists x\, \exists y\ (x \to^n y) \wedge t_i(x) \wedge t_j(y)\,.$$

Without wildcard we cannot express $\to^n$, but we can circumvent this obstacle using $\overset{+}{\to}$ if we modify our encoding properly. We encode the tile t_i as the word

$$w_i = \triangleright \bar{a} a^i b a^j \bar{a} \triangleleft$$

with $\bar{a} = aa^k$ and $i + j + 1 = k$. For the DTD we take $r \to w_0 W^* w_k$, where $W = \{w_i \mid i = 1, 2, \ldots, k\}$. The patterns are replaced with

$$x_1 \xrightarrow{w_i} x_1' \to x_2 \xrightarrow{w_j} x_2'\,,$$

$$x_1 \xrightarrow{w_i} x_1' \to x_2 \xrightarrow{w_*} x_2' \to \ldots \to x_n \xrightarrow{w_*} x_n' \to x_{n+1} \xrightarrow{w_j} x_{n+1}'\,,$$

where $x \xrightarrow{w_i} x'$ is a pattern that says that the segment of the word from position x to x' is w_i and $x \xrightarrow{w_*} y$ is the following pattern (see also Fig. 2)

$$(x \xrightarrow{\triangleright\bar{a}} x'') \wedge (x' \xrightarrow{\bar{a}b\bar{a}} y') \wedge (y'' \xrightarrow{\bar{a}\triangleleft} y) \wedge (x \overset{+}{\to} x' \overset{+}{\to} x'') \wedge (y'' \overset{+}{\to} y' \overset{+}{\to} y)\,. \qquad \square$$

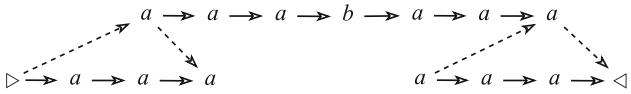

Fig. 2. Pattern $x \xrightarrow{w_*} y$ for $k = 2$

Thus BC-SAT($\Rightarrow$) is PSPACE-complete if either wildcard or $\xrightarrow{+}$ and arbitrary joins are allowed. If we forbid wildcards and restrict the use of joins, the complexity drops to NP. Using $\textsc{Match}_\pi$ and the following observation, we prove the upper bound for a relatively large class of patterns, extending tree patterns.

Lemma 3. *For all $w_1, w_2, \ldots, w_n \in \Gamma^*$ there is a linear-size deterministic automaton recognizing $\bigcup_{i=1}^{n} \Gamma^* w_i \Gamma^*$. One can compute it in* PTIME.

Proof. States of the automaton are prefixes of w_i's. After reading u, the state is the longest prefix that is a suffix of u. If the prefix is the whole word w_i, the automaton moves to a distinguished accepting state. □

Theorem 5. *For patterns whose firm sub-patterns do not contain $\xrightarrow{+}$, the problem* BC-SAT($\rightarrow, \xrightarrow{+}$) *on words is* NP*-complete.*

Proof. The lower bound is proved in [9]. To get the upper bound, we prove a polynomial model property. Given that $\Rightarrow$-patterns can be evaluated in PTIME, the proposition follows.

Let $w \in L(\mathcal{A})$ be a word satisfying a Boolean combination φ. For each pattern π in φ consider its *earliest partial matching*, i.e., the partial matching computed by $\textsc{Match}_\pi$. Clearly, π is satisfied if and only if its earliest partial matching is total. It suffices to show that the segments of w outside of the partial matches can be chosen small, without changing the matches.

Suppose that $w = u_1 v u_2$ and v is not touched by the partial matchings. A partial matching is earliest if and only if each firm sub-pattern π_0 is matched at its first occurrence after the *launching point*: the latest position i such that matching π_0 at i (regardless of labels) violates some $\xrightarrow{+}$ edge entering π_0. When shortening v we only need to make sure that we do not introduce an occurrence of a subpattern between its launching point and its original match in w. For sub-patterns matched in u_1 changing v makes no difference. Suppose π_0 is matched in u_2. Where can the launching point of π_0 be? If it is enforced by a sub-pattern matched in u_1, it is in u_1. If it is enforced by a subpattern matched in u_2, it is either in u_2 or within the last $|\pi_0|$ positions of v.

Let $\pi_1, \pi_2, \ldots, \pi_k$ be all sub-patterns matched in u_2 whose launching points are in u_1. Since they contain no $\xrightarrow{+}$ nor $\_$, they can be turned into single words by merging along the $\rightarrow$ edges. Let $\mathcal{B}$ be the deterministic automaton accepting words that contain some π_i (Lemma 3). Let $v = v_1 v' v_2$, where $|v_1| = |v_2|$ is equal to the maximal size of a firm subpattern. By standard pumping we can shorten v' to at most $\|\mathcal{A}\| \cdot \|\mathcal{B}\|$, without introducing new occurrences of π_i's in

vu_2. Since we are not touching v_1, we do not introduce new occurrences of π_i's in the whole word w. Similarly, since we are not touching v_2, the patterns whose launching points are in v_2u_2 are not influenced either. □

5 Back to Trees

We now lift the restriction on models and see what happens for trees. We have already seen that BC-SAT for full CQs is exponentially harder than for tree patterns: 2ExpTime versus ExpTime, and ExpSpace versus PSpace under non-recursive schemas (Theorems 1–3). Here we consider several restrictions on CQs and schemas that lower the complexity of CQs to that of tree patterns.

We show first that, for vertical tree patterns extended with arbitrary horizontal CQs over siblings, our PSpace algorithm for BC-SAT on words can be incorporated into the procedures for tree patterns without increasing their complexity. (Allowing joins with arbitrary horizontal CQs would immediately violate the intended tree structure of the vertical part of the pattern.) We say that a pattern is *forest-like* if its $\Downarrow$-subgraph is a disjoint union of trees and all vertical edges coming to the same connected $\Rightarrow$-subpattern originate in the same vertex.

Theorem 6. *For forest-like patterns,* BC-SAT$(\Downarrow, \Rightarrow)$ *is* ExpTime*-complete, and under non-recursive schemas it is* PSpace*-complete.*

Proof. For a forest-like pattern π we shall construct an equivalent deterministic automaton $\mathcal{A}_\pi$, whose states and transitions can be generated in PSpace. (Recall that a tree automaton is (bottom-up) deterministic, if for all $q_1, q_2 \in Q$ and $a \in \Sigma$ there exists exactly one state q such that $(q_1, q_2, a, q) \in \delta$.) Using this construction one can reduce BC-SAT$(\Downarrow, \Rightarrow)$ to nonemptiness of tree automata in PSpace. Both upper bounds follow, since nonemptiness of $\mathcal{A}$ over trees of depth d can be tested in space $\mathcal{O}(d \cdot \log \|\mathcal{A}\|)$, and over arbitrary trees in PTime.

A *horizontal component* of π is a connected component of the $\Rightarrow$-subgraph of π. Let $H_\pi = \langle V_\pi, \downarrow, \downarrow^+ \rangle$ be a graph over horizontal components of π, where edge $\pi_1 \downarrow \pi_2$ is present if $x \downarrow y$ for some $x \in \pi_1$ and $y \in \pi_2$, and $\pi_1 \downarrow^+ \pi_2$ is present if $x \downarrow^+ y$ for some $x \in \pi_1$ and $y \in \pi_2$, but there is no edge $\pi_1 \downarrow \pi_2$. Since π is forest-like, this graph is a forest. The subtree of H_π rooted at π_1 defines a subpattern of π, denoted by $(\pi_1)_\Downarrow$. We call such subpatterns *subtrees of* π.

The automaton $\mathcal{A}_\pi$, after reading the sequence of children of a node v, passes to v information about subtrees of π that were matched in the children of v and those that were matched in the children of some descendant of v. The automaton accepts, if the information passed from the root says that π was matched. To compute the information to be passed to v the automaton needs to aggregate the information passed from v's grandchildren to their parents. This is done by a modified version of Match working over an extended alphabet, described below.

A subtree $(\pi_1)_\Downarrow$ of π can be viewed as a horizontal pattern obtained from π_1 by including in the label of each vertex x the information about the subtrees of π to which x is connected by $\downarrow$ and $\downarrow^+$ edges. At each step Match is fed with a symbol that consists of the label of a tree node u and the information passed to

u from its sequence of children. (At the leaf level of T this information is void and MATCH works just like for words.) MATCH is only altered in this way that a vertex labelled with an extended label σ can be matched in a position labelled with an extended label τ if the original labels agree and all patterns listed in σ are also listed in τ (keeping the distinction between patterns connected by $\downarrow$ and $\downarrow^+$). It is straightforward to check that this does not influence correctness of MATCH. Observe that the extended alphabet is exponential, but each symbol can be stored in polynomial memory. Hence, MATCH still works in memory polynomial in the size of the pattern.

This procedure can be easily implemented by an exponential deterministic tree automaton. Within a sequence of children, $\mathcal{A}_\pi$ behaves like the automaton implementing MATCH($\tilde{\pi}$), where $\tilde{\pi}$ is the disjoint union of all subtrees of π. It reads the extended label from the label of the current child u and the state coming from the children of u. When the last child is read, the information about matched subtrees of π is complete and can be passed up, to the parent. □

From this result we obtain further upper bounds. For purely horizontal patterns, a standard pumping argument allows us to bound the height of the witnessing tree by the size of the schema, and with some care the algorithm for non-recursive schemas can be used even if the original schema is recursive.

Corollary 1. BC-SAT($\Rightarrow$) *is* PSPACE*-complete.*

Furthermore, as observed in [16], each pattern using no $\downarrow^+$ can be turned in PTIME into an equivalent forest-like pattern by simply merging each pair of vertices that have outgoing $\downarrow$ edges to the same connected $\Rightarrow$-subpattern. Hence, we immediately get the following corollary (hardness from Theorem 1).

Corollary 2. BC-SAT($\downarrow, \Rightarrow$) *is* EXPTIME*-complete and* PSPACE*-complete under non-recursive schemas.*

Finally, with a little more effort one can prove that in the presence of a non-recursive DTD the same holds for patterns that do not use wildcard.

Corollary 3. BC-SAT($\downarrow, \downarrow^+, \rightarrow, \overset{+}{\rightarrow}$) *is* PSPACE*-complete under non-rec. DTDs.*

The lower bound follows from Theorem 1 and the upper bound relies on the fact that in the presence of a non-recursive DTD labels come in a fixed order in the paths. For non-recursive tree automata this is no longer the case. In fact, under such schemas one can carry over the lower bounds of Theorem 3 to the case without wildcard.

6 Disjunction-Free DTDs

Theorem 3 shows that under non-recursive schemas BC-SAT does not get much easier. We now introduce another restriction, often used in combination with non-recursivity in complex data management tasks [2, 3]: we limit the use of

disjunction. A DTD is *disjunction-free* if its regular expressions use only concatenation, Kleene star and the operator $\alpha^{\leq m} = (\varepsilon \mid \alpha \mid \alpha^2 \mid \ldots \mid \alpha^m)$.

BC-SAT under disjunction-free DTDs is not easier unless $\rightarrow$ is forbidden. Indeed, using $\rightarrow$ we can simulate full DTDs, e.g., a production $a \rightarrow \alpha \mid \beta$ can be simulated by $a \rightarrow \sharp(\triangleright\alpha\triangleleft)^*(\triangleright\beta\triangleleft)^*\sharp$, with conjunct $\neg\exists x\, \exists y \left(\sharp(x) \rightarrow \sharp(y)\right) \wedge \neg\exists x\, \exists y \left(\triangleleft(x) \rightarrow \triangleright(y)\right)$ added to the combination tested for satisfiability.

If $\rightarrow$ is forbidden, the complexity under disjunction-free non-recursive DTDs drops to low levels of the polynomial hierarchy, compared to NExpTime for non-recursive DTDs allowing disjunction (Theorem 3).

Theorem 7. *Under non-recursive disjunction-free DTDs* $\mathsf{BC\text{-}SAT}(\Downarrow, \overset{+}{\rightarrow})$ *is* Σ_2P*-complete and* NP*-complete for tree patterns.*

The problem is Σ_2P-hard already for Boolean combinations of the form $\pi_1 \wedge \neg\pi_2$ where π_1 is a pattern with a single node, but without π_1 it is coNP-complete.

If the non-recursivity restriction is lifted the complexity is still (potentially) below the general 2ExpTime lower bound.

Theorem 8. *Under disjunction-free DTDs* $\mathsf{BC\text{-}SAT}(\Downarrow, \overset{+}{\rightarrow})$ *is in* NExpTime *and* PSpace*-complete for tree patterns. The lower bound holds already for containment of unions of tree patterns using only* $\downarrow, \_$.

7 Conclusions

We have shown that under several independent restrictions, CQs have the same complexity of the satisfiability of Boolean combinations, and the containment of unions of queries problem, as tree patterns. Most importantly, vertical tree patterns can be extended with full horizontal CQs over children without increasing the complexity of static analysis tasks. We have also showed that under non-recursive, disjunction-free schemas the complexity of static analysis for CQs without the next-sibling relation is in low levels of the polynomial hierarchy. This could be applied in the analysis of mappings between nested-relational schemas [2]. (We point out the complexity gap for general disjunction-free schemas as an elegant theoretical challenge.) We focused on containment of UCQs, since this is the problem relevant for XML metadata management, but a finer analysis of the containment for CQs would also be desired (the 2ExpTime-lower bound of [9] holds already for validity of CQs). Similarly, patterns with data comparisons might be considered (again, some cases are settled in [9]).

Acknowledgements. This work is part of the *Querying and Managing Navigational Databases* project realized within the Homing Plus programme of the Foundation for Polish Science, co-financed by the European Union from the Regional Development Fund within the Operational Programme Innovative Economy ("Grants for Innovation"). We thank Claire David for inspiring discussions and careful reading of a preliminary version of this paper, and the anonymous referees for helpful comments motivating us to improve the presentation of the paper.

References

1. Amano, S., David, C., Libkin, L., Murlak, F.: On the Tradeoff between Mapping and Querying Power in XML Data Exchange. In: ICDT, pp. 155–164 (2010)
2. Amano, S., Libkin, L., Murlak, F.: XML schema mapping. In: PODS, pp. 33–42 (2009)
3. Arenas, M., Libkin, L.: XML data exchange: consistency and query answering. J. ACM 55(2) (2008)
4. Benedikt, M., Bourhis, P., Senellart, P.: Monadic Datalog Containment. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 79–91. Springer, Heidelberg (2012)
5. Benedikt, M., Fan, W., Geerts, F.: XPath satisfiability in the presence of DTDs. J. ACM 55(2) (2008)
6. Bex, G.J., Neven, F., Van den Bussche, J.: DTDs versus XML Schema: a practical study. In: WebDB, pp. 79–84 (2004)
7. Björklund, H., Gelade, W., Martens, W.: Incremental XPath evaluation. ACM Trans. Database Syst. 35(4), 29 (2010)
8. Björklund, H., Martens, W., Schwentick, T.: Conjunctive Query Containment over Trees. In: Arenas, M. (ed.) DBPL 2007. LNCS, vol. 4797, pp. 66–80. Springer, Heidelberg (2007)
9. Björklund, H., Martens, W., Schwentick, T.: Optimizing Conjunctive Queries over Trees Using Schema Information. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 132–143. Springer, Heidelberg (2008)
10. Bojańczyk, M., Kołodziejczyk, L.A., Murlak, F.: Solutions in XML data exchange. In: ICDT, pp. 102–113 (2011)
11. ten Cate, B., Lutz, C.: The Complexity of Query Containment in Expressive Fragments of XPath 2.0. J. ACM 56(6), 1–48
12. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: STOC, pp. 77–90 (1977)
13. David, C.: Complexity of Data Tree Patterns over XML Documents. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 278–289. Springer, Heidelberg (2008)
14. David, C., Libkin, L., Murlak, F.: Certain answers for XML queries. In: PODS, pp. 191–202 (2010)
15. Fagin, R., Kolaitis, P., Miller, R., Popa, L.: Data exchange: semantics and query answering. Theor. Comp. S. 336, 89–124 (2005)
16. Gottlob, G., Koch, C., Schulz, K.: Conjunctive queries over trees. J. ACM 53, 238–272 (2006)
17. Hidders, J.: Satisfiability of XPath Expressions. In: Lausen, G., Suciu, D. (eds.) DBPL 2003. LNCS, vol. 2921, pp. 21–36. Springer, Heidelberg (2004)
18. Ishihara, Y., Morimoto, T., Shimizu, S., Hashimoto, K., Fujiwara, T.: A Tractable Subclass of DTDs for XPath Satisfiability with Sibling Axes. In: Gardner, P., Geerts, F. (eds.) DBPL 2009. LNCS, vol. 5708, pp. 68–83. Springer, Heidelberg (2009)
19. Lenzerini, M.: Data integration: a theoretical perspective. In: PODS, pp. 233–246 (2002)
20. Marx, M.: XPath with Conditional Axis Relations. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 477–494. Springer, Heidelberg (2004)
21. Miklau, G., Suciu, M.: Containment and equivalence for a fragment of XPath. J. ACM 51(1), 2–45 (2004)

22. Murata, M., Lee, D., Mani, M., Kawaguchi, K.: Taxonomy of XML schema languages using formal language theory. ACM Transactions on Internet Technology 5(4), 1–45 (2005)
23. Neven, F.: Automata Theory for XML Researchers. SIGMOD Record 31(3), 39–46 (2002)
24. Neven, F., Schwentick, T.: On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. Logical Meth. in Comp. Sci. 2(3), 1–30 (2006)
25. Wood, P.T.: Containment for XPath Fragments under DTD Constraints. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 297–311. Springer, Heidelberg (2002)