Validity of Tree Pattern Queries With Respect to Schema Information

Henrik Björklund[1], Wim Martens[2], and Thomas Schwentick[3]

[1] Umeå University, Sweden
[2] University of Bayreuth, Germany
[3] TU Dortmund University, Germany

Abstract. We prove that various containment and validity problems for tree pattern queries with respect to a schema are EXPTIME-complete. When one does not require the root of a tree pattern query to match the root of a tree, validity of a non-branching tree pattern query with respect to a Relax NG schema or W3C XML Schema is already EXPTIME-hard when the query does not branch and uses only child axes. These hardness results already hold when the alphabet size is fixed. Validity with respect to a DTD is proved to be EXPTIME-hard already when the query only uses child axes and is allowed to branch only once.

1 Introduction

Tree pattern queries are omnipresent in query and schema languages for XML. They form a logical core of the query languages XPath, XQuery, and XSLT, and they are needed to define key constraints in XML Schema. Static analysis problems such as containment, satisfiability, validity, and minimization for tree pattern queries have been studied for over a decade [16, 10, 18, 3] since their understanding helps us, for example, in the development of query optimization procedures. Since queries can usually be optimized more if schema information is taken into account, these static analysis problems are also relevant in settings with schema information [18, 3]. This is the setting that we consider.

The literature uses the term "tree pattern query" for a variety of query languages. In this paper, we use the tree pattern queries as in [16], which can use labels, wildcards (*), the child relation (/), the descendant relation (//), and filtering ([$\cdot$]) which allows them to branch. In the following, we us the terms *path query* for tree pattern queries without [$\cdot$] and *child-only query* for tree pattern queries without //. Containment, satisfiability, and validity of tree pattern queries are closely related to each other in the usual way, i.e., satisfiability and validity are special cases of containment. Since tree pattern queries are not closed under the Boolean operations, satisfiability and validity often have a lower complexity than containment. Taking schema information into account usually increases the computational complexity. For example, containment of tree pattern queries is coNP-complete [16] but becomes EXPTIME-complete if schema information, even in its weakest form (a DTD) is provided [18].[4]

We investigate the complexity of the validity problem (with schema information) and obtain complexity lower bounds that contrast rather sharply with known upper bounds. Hashimoto et al. [12] showed that validity of path queries with respect to DTDs is in PTIME. We prove:

- Validity of path queries with respect to tree automata is EXPTIME-hard, even if the tree automata are XSDs with a constant-size alphabet (Theorems 10 and 11).
- Validity of child-only tree pattern queries with respect to DTDs is already EXPTIME-hard even if the tree pattern queries branch only once and the branch has only one node (Theorem 12).
- As a simpler application of our techniques we prove as a warm-up: inclusion of a DFA in a regular expression of the form $\Sigma^* a \Sigma^n b \Sigma^*$ is PSPACE-complete over $\Sigma = \{a, b, c\}$ (Theorem 9). This means that validity of very simple child-only path queries is PSPACE-hard, even if trees don't branch.[5]

Each case is only a very slight extension of the above mentioned PTIME scenario of Hashimoto et al. [12]. Our semantics of path and tree pattern queries is such that the root of the query does not need to be matched by the root of the tree. For our EXPTIME-hardness results to hold when using the more restricted semantics of [12], we would need queries to have one additional descendant axis, placed at the root. On the other hand, the PTIME upper bound of [12] also holds in our setting.

Our lower bounds are also relevant in terms of conjunctive queries over trees. For example, Benedikt et al. ([2], Corollary 3) proved a matching EXPTIME upper bound for validity of UCQs (Unions of Conjunctive Queries) with respect to a tree automaton. Here, UCQs form a class of queries that do not use the descendant axis but are strictly more general than child-only tree pattern queries since their syntactic structure is not required to be tree-shaped. Recently, static analysis for such queries (with schema information) has also been investigated in [5, 17], with complexity results ranging from tractable to 2EXPTIME-complete.

In our proofs we use restricted variants of tiling games (Section 3) that may be interesting in their own right.

2 Preliminaries

We use standard definitions and notation for regular expressions, DFAs and NFAs, to be found in the appendix.

[4] Schemas can be given as DTDs (the weakest form), XSDs (in the middle), or tree automata (the strongest form; defining regular tree languages), see [15].

[5] This result has already been used in the context of XML key inference [1].

Trees and Tree Pattern Queries. Schema languages for XML recognize trees which are rooted, ordered, finite, labeled, unranked, and directed from the root downwards. For this reason, we consider finite trees in which nodes can have arbitrarily many children, ordered from left to right. However, we note that the results in this paper hold equally well for automata and DTDs recognizing unordered trees, that is, trees in which the children can occur in any order. More formally, we view a tree t as a relational structure over a finite number of unary labeling relations $a(\cdot)$, for $a \in \Sigma$, and binary relations $\mathit{Child}(\cdot,\cdot)$ and $\mathit{NextSibling}(\cdot,\cdot)$. Here, $a(u)$ expresses that u is a node with label a, and $\mathit{Child}(u,v)$ (respectively, $\mathit{NextSibling}(u,v)$) expresses that v is a child (respectively, the right sibling) of u. We denote the set of nodes of a tree t by $\mathrm{Nodes}(t)$. We assume that trees are non-empty, i.e., $\mathrm{Nodes}(t) \neq \emptyset$. By $\mathrm{Edges}(t)$ we denote the set of child edges of t. For a node u, we denote by $\mathrm{lab}^t(u)$ the unique symbol a such that $a(u)$ holds in t. We often omit t from this notation when t is clear from the context. By $\mathrm{root}(t)$ we denote the root node of t. For a node u of t, we denote by $\text{anc-str}^t(u)$ the string obtained by concatenating all labels on the path from the root of t to u. That is, $\text{anc-str}^t(u) = \mathrm{lab}^t(u_1)\cdots\mathrm{lab}^t(u_k)$ where $u_1 = \mathrm{root}(t)$, $u_k = u$, and $u_1 \cdots u_k$ is the path from u_1 to u_k. Similarly, $\text{ch-str}^t(u)$ is the concatenation of the labels of all children of u, from left to right.

Definition 1 [Tree Pattern Query] A *tree pattern query*, (*TPQ*), over Σ is a tuple $T = (p, \mathrm{Anc})$, where p is a tree that uses the labeling alphabet $\Sigma \uplus \{*\}$ and $\mathrm{Anc} \subseteq \mathrm{Edges}(t)$ is the set of *ancestor edges.*

Here, we use $*$ as a wildcard label. More formally, the semantics of TPQs is defined as follows. Let $T = (p, \mathrm{Anc})$ be a TPQ and let s be a tree. Let $v_p \in \mathrm{Nodes}(p)$ and $v_s \in \mathrm{Nodes}(s)$. We say that v_s *matches* v_p if either $\mathrm{lab}(v_p) = *$ or $\mathrm{lab}(v_s) = \mathrm{lab}(v_p)$. An *embedding* of $T = (p, \mathrm{Anc})$ on a tree s is a mapping m from $\mathrm{Nodes}(p)$ to $\mathrm{Nodes}(s)$ such that,

- for every node $v \in \mathrm{Nodes}(p)$, $m(v)$ matches v, and
- for every two nodes $v_1, v_2 \in \mathrm{Nodes}(p)$,
  - if $(v_1, v_2) \in \mathrm{Edges}(p) \setminus \mathrm{Anc}$, then $(m(v_1), m(v_2)) \in \mathrm{Edges}(s)$;
  - if $(v_1, v_2) \in \mathrm{Anc}$, then $m(v_1)$ is an ancestor of $m(v_2)$ in s.

Notice that the root of p does not need to be mapped to the root of s, which is important when comparing our results to related work. The *language* defined by T is denoted $L(T)$ and consists of all trees s for which there is an embedding of T into s. Notice that our semantics defines tree pattern queries as Boolean queries.

Tree pattern queries form a natural fragment of the XPath query language [8]. We assume familiarity with the standard XPath notation of tree pattern queries (see, e.g., [16]). Figure 1 contains an example of a tree pattern query and its corresponding XPath notation.

Schemas. We introduce our abstractions of Document Type Definition (DTD) [6], XML Schema [20], and Relax NG schemas [9].

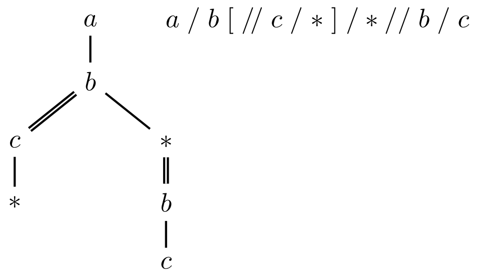

Fig. 1. A tree pattern query $T = (p, \text{Anc})$ depicted as a tree (on the left) and in XPath notation (on the right). On the left, edges in Anc are drawn as double lines. On the right, the bracketed part corresponds to the left branch in the tree. Slashes ('/') represent edges and double slashes ('//') represent edges in Anc.

Definition 2 A *Document Type Definition (DTD)* over Σ is a triple $D = (\Sigma, d, S)$ where $S \subseteq \Sigma$ is the set[6] of start symbols and d is a set of rules of the form $a \rightarrow R$, where $a \in \Sigma$ and R is a regular expression over Σ. No two rules have the same left-hand side.

A tree t *satisfies* D if *(i)* $\text{lab}^t(\text{root}(t)) \in S$ and, *(ii)* for every $u \in \text{Nodes}(t)$ with label a and n children $u_1, \ldots, u_n$ from left to right, there is a rule $a \rightarrow R$ in d such that $\text{lab}^t(u_1) \cdots \text{lab}^t(u_n) \in L(R)$. By $L(D)$ we denote the set of trees satisfying D.

We abstract XML Schema Definitions as *DFA-based XSDs*. DFA-based XSDs were introduced by Martens, Neven, Schwentick, and Bex [15, 14] as formal model for XML Schema convenient in proofs.[7]

Definition 3 A *DFA-based XSD* is a pair (A, λ), where $A = (Q, \Sigma, \delta, \{q_{\text{init}}\}, \emptyset)$ is a DFA with initial state q_{init} and λ is a function mapping each state in $Q \setminus \{q_{\text{init}}\}$ to a regular expression over Σ.

An tree t *satisfies* (A, λ) if, for every node u, $A(\text{anc-str}^t(u)) = \{q\}$ implies that $\text{ch-str}^t(u)$ is in the language defined by $\lambda(q)$.

We abstract from Relax NG schemas [9] by unranked tree automata.

Definition 4 A *nondeterministic (unranked) tree automaton (NTA)* over Σ is a quadruple $A = (Q, \Sigma, \delta, F)$, where Q is a finite set of states, $F \subseteq Q$ is the set of accepting states, and δ is a set of transition rules of the form $(q, a) \rightarrow L$, where $q \in Q$, $a \in \Sigma$, and L is a regular string language over Q, represented by a regular expression.[8]

[6] DTDs usually have a single start symbol in the literature. Our abstraction is slightly closer to reality; it has no influence on our complexity results.

[7] XML Schema Definitions are sometimes also abstracted as single-type EDTDs, but it is well-known that DFA-based XSDs and single-type EDTDs can be converted back and forth in polynomial time [11]. DFA-based XSDs [14] are called DFA-based *DTDs* in [11] but are the same thing. Since they are a formal model for XSDs, we choose to reflect this in their name.

[8] For our complexity results, it does not matter whether the languages L are represented by regular expressions, nondeterministic string automata, deterministic string automata, or even as a finite set of strings.

A *run* of A on a tree t is a labeling $r : \text{Nodes}(t) \to Q$ such that, for every $u \in \text{Nodes}(t)$ with label a and children $u_1, \ldots, u_n$ from left to right, there exists a rule $(q, a) \to L$ such that $r(u) = q$ and $r(u_1) \cdots r(u_n) \in L$. Note that when u has no children, the criterion reduces to $\varepsilon \in L$, where ε denotes the empty string. A run on t is *accepting* if the root of t is labeled with an accepting state, that is, $r(\text{root}(t)) \in F$. A tree t is *accepted* if there is an accepting run of A on t. The set of all accepted trees is denoted by $L(A)$ and is called a *regular tree language*. From now on, we use the word "schema" to refer to DTDs, DFA-based XSDs, or NTAs.

It is well-known that DTDs are less expressive than DFA-based XSDs, which in turn are less expressive than NTAs [15]. Likewise, DTDs can be polynomial-time converted into DFA-based XSDs, which can be polynomial-time converted into NTAs.

We are concerned with the following decision problem:

Definition 5 *Validity w.r.t. a schema*: Given a TPQ T and a schema S, is $L(S) \subseteq L(T)$?

3 Tiling Problems and Games

We recall definitions and properties of tiling systems, corridor tilings, and their associated games. We define a restricted form of corridor tiling games that remains EXPTIME-complete and may be of interest in its own right.

A *tiling system* $S = (T, V, H, t_{\text{fin}})$ consists of a finite set T of *tiles*, two sets $V, H \subseteq T \times T$ of *vertical* and *horizontal constraints*, respectively, and a *final tile* $t_{\text{fin}} \in T$. A *solution* for a tiling system S is a mapping $\tau : \{1, \ldots, n\} \times \{1, \ldots, m\} \to T$ for some $n, m \geq 2$ such that (i) the horizontal constraints are fulfilled, that is, for every $i \in \{1, \ldots, n-1\}, j \in \{1, \ldots, m\}$: $(\tau(i, j), \tau(i+1, j)) \in H$; (ii) the vertical constraints are fulfilled, that is, for every $i \in \{1, \ldots, n\}, j \in \{1, \ldots, m-1\}$: $(\tau(i, j), \tau(i, j+1)) \in V$; and (iii) the final tile is correct, that is, $\tau(n, m) = t_{\text{fin}}$.

In the *corridor tiling* problem one is given a tiling system S and a word $w = w_1 \cdots w_n \in T^*$ of tiles, called the *initial row*. The problem asks whether there exists a solution to S with bottom row w, that is a mapping $\tau : \{1, \ldots, n\} \times \{1, \ldots, m\} \to T$ as above with $n = |w|$ such that $\tau(i, 1) = w_i$ for every $i \in \{1, \ldots, n\}$.

It is well-known that the corridor tiling problem is PSPACE-complete [7]. However, this result even holds for some fixed tiling systems S. For a tiling system S, we write Tiling(S) for the set of strings w such that S has a solution with initial row w.

Theorem 6 ([7], Section 4) *There is a tiling system S such that Tiling(S) is PSPACE-hard.*

This strengthening can be obtained by applying the argument of Section 4 in [7] to some fixed PSPACE-complete language L and a TM M for L.

Tiling systems can also be used to define two-player games. The input for a tiling game is the same as for the corridor tiling problem but the underlying idea is different: two players, CONSTRUCTOR and SPOILER, alternatingly choose tiles. CONSTRUCTOR's goal is to build a solution for the tiling system and SPOILER's goal is to prevent that.

More formally, we associate with a tiling system S and an initial row w for S a 2-player game as follows. The word w induces a mapping $\tau : \{1, \ldots, n\} \times \{1\} \to T$, where $n = |w|$. The two players alternatingly choose tiles $t \in T$, implicitly defining $\tau(1,2), \tau(2,2), \ldots, \tau(n,2), \tau(1,3)$, etc. A move is *legal* if it satisfies the constraints. More precisely, a tile t is a legal move as $\tau(i,j)$ if $(\tau(i-1,j), \tau(i,j)) \in H$ and $(\tau(i,j-1), \tau(i,j)) \in V$. Players are not allowed to play a non-legal move. CONSTRUCTOR loses the game if, at any point, one of the players cannot make a legal move. On the other hand, CONSTRUCTOR wins if at some point a correct corridor tiling for S is constructed (for some m).

For a tiling system S, we denote by TilingWinner(S) the set of all strings w such that CONSTRUCTOR has a winning strategy for the game induced by S and w. From [7] the following theorem immediately follows.[9]

Theorem 7 ([7], Theorem 5.1)
(a) For every tiling system S, TilingWinner(S) $\in$ EXPTIME, and
(b) there is a tiling system S, for which TilingWinner(S) is EXPTIME-hard.

For our reductions we need to work with suitably restricted tiling systems which we define next. Given a system S and an initial row w, a *valid rectangle* for S and w is a tiling that is a solution except that the last tile need not be t_{fin}, i.e., it is a mapping $R : \{1, \ldots, n\} \times \{1, \ldots, m\} \to T$ with initial row w, where $n = |w|$ and $m \geq 1$ that respects V and H. A *tiling prefix* for S and w is a valid rectangle plus the beginning of a next row, that is, a mapping P from $\{1, \ldots, n\} \times \{1, \ldots, m\} \cup \{1, \ldots, i\} \times \{m+1\}$ to T, for some $i \in \{1, \ldots, n\}$ and $m \geq 1$, with bottom row w that respects V and H. A tiling prefix for S and w is *valid* if the partial row can be completed to form a valid rectangle. We define the *length* of P to be $nm + i$. In particular, every valid rectangle is also a valid prefix. Given a tiling prefix P and a tile t, we write $P.t$ for the extension of P by t.

We call a tiling system $S = (T, V, H, t_{\text{fin}})$ *restricted* if the following holds for every initial row w.

(1) If $|w|$ is odd, then $w \notin$ Tiling(S).
(2) For every valid prefix P there are exactly two tiles t_1 and t_2 such that $P.t_1$ and $P.t_2$ are valid prefixes.
(3) For every odd length valid prefix P, there are exactly two tiles t_1 and t_2 such that $P.t_1$ and $P.t_2$ are tiling prefixes.

The restriction guarantees that, if CONSTRUCTOR has a winning strategy, she has one in which SPOILER always has exactly two legal moves.

[9] Similarly as for Tiling(S), it suffices to fix an ATM for some EXPTIME-complete language in the proof to infer Theorem 7 (b) from Theorem 5.1 in [7].

Proposition 8 *There is a restricted tiling system S, for which TilingWinner(S) is EXPTIME-hard.*

4 String Languages

Theorem 9 *Validity of regular expressions w.r.t. a DFA is PSPACE-complete even for regular expressions*[10] *of the form $\Sigma^* a \Sigma^n b \Sigma^*$ and DFAs over the alphabet $\Sigma = \{a, b, c\}$.*

Proof. Obviously, the problem is in PSPACE since it can be reduced in logarithmic space to the containment problem for NFAs, which is known to be PSPACE-complete [19]. We show the lower bound by reduction from Tiling(S), i.e., corridor tiling with a fixed tiling system. Let $S = (T, V, H, t_{\text{fin}})$ with $T = \{t_1, \dots, t_k\}$ be a tiling system such that Tiling(S) is PSPACE-hard. Notice that S exists by Theorem 6.

We associate with every tiling function $\tau : \{1, \dots, n\} \times \{1, \dots, m\} \to T$ a string w_τ by simply concatenating all tiles in row-major order. More precisely, w_τ is the string from T^* of length mn that carries at position $(j-1)n + i$ tile $\tau(i, j)$, for every $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$.

From S and an initial row w a DFA $A(S, w)$ can be defined that tests whether a word $v \in T^*$ has the following properties: (i) v has prefix w; (ii) the length of v is a multiple of $n \overset{\text{def}}{=} |w|$; (iii) the tiling function $\tau : \{1, \dots, n\} \times \{1, \dots, m\} \to T$ corresponding to v fulfills the horizontal constraints; and (iv) $\tau(n, m) = t_{\text{fin}}$. We show in the appendix how we can construct $A(S, w)$ polynomial time.

However, for the actual reduction we use a more elaborate encoding of tiles and define the DFA accordingly. We simultaneously encode tiles and their relevant vertical constraints as strings of length $2k$. For each $i, j \in \{1, \dots, k\}$ we let e_{ij} be a symbol that encodes whether $(t_i, t_j) \in V$ as follows.

$$e_{ij} \overset{\text{def}}{=} \begin{cases} a & \text{if } (t_i, t_j) \notin V \\ c & \text{otherwise.} \end{cases}$$

Then, for each $i \in \{1, \dots, k\}$, we encode tile t_i as the string

$$\text{enc}(t_i) \overset{\text{def}}{=} cc \cdots cbc \cdots ce_{i1} \cdots e_{ik}$$

of length $2k$ in which the entry labeled b is at position i. For a string $v \in T^*$, we write enc(v) for the symbol-wise encoding of v. It is straightforward to construct from $A(S, w)$ an automaton $A'(S, w)$ that accepts all encodings enc(v) of strings $v \in L(A(S, w))$ (see Appendix). Finally, the regular expression q_w is just $(a+b+c)^* a (a+b+c)^{(2n-1)k-1} b (a+b+c)^*$. We prove in the appendix that the reduction is correct. □

5 Hardness Results on Trees

In this section we are going to prove the following three results.

[10] Σ^n abbreviates concatenations of n symbols from Σ.

Theorem 10 *Validity of tree pattern queries w.r.t. an NTA is EXPTIME-complete even for path queries of the form $a/*/*/\cdots/*/b$ over schemas with three symbols.*

Theorem 11 *Validity of tree pattern queries w.r.t. a DFA-based XSD is EXPTIME-complete even for path queries of the form $a/*/*/\cdots/*/b$ over schemas with four symbols.*

Theorem 12 *Validity of tree pattern queries w.r.t. a DTD is EXPTIME-complete even for tree pattern queries of the form $*[/a]/*/\cdots/*/b$ over DTDs.*

Notice the subtle differences between the three cases: In the NTA case it suffices to have path queries and three alphabet symbols. For DFA-based XSDs we use one more alphabet symbol due to their limited expressiveness when compared to NTAs. If we limit the expressiveness even more to DTDs, then validity of path queries is not hard anymore, as was shown by Hashimoto et al. [12].

Theorem 13 ([12], Theorem 3) *Validity of path queries w.r.t. a DTD is in PTIME.*

However, even allowing the path to have one additional leaf branching off makes the validity problem EXPTIME-hard, even w.r.t. DTDs. Thus, compared to Theorem 13, allowing a single branching as opposed to a pure path query or using DFA-based XSDs as opposed to DTDs results in a provably exponential blow-up in the time complexity for Validity.

All the problems considered in this section are in EXPTIME because of the following result.

Theorem 14 *Validity of tree pattern queries query w.r.t. an NTA is in EXPTIME.*

The lower bounds in Theorems 11 and 12 are shown by reductions from the problem of identifying the winner in a 2-player corridor tiling game. For the purpose of these reductions we use the restricted form of tiling games from Section 3.

Strategies for Constructor for some tiling system S and initial row w can be represented by *strategy trees* as usual. The nodes of such a tree carry the tiles chosen in the game. Each node that corresponds to a tile chosen by Spoiler has a child labelled with the symbol that is chosen according to Constructor's strategy. Each node v that corresponds to a tile chosen by Constructor has one child for every possible legal move by Spoiler. Every internal node in the tree corresponds to a tiling prefix, induced by the path from the root to that node.

Proof (of Theorem 11). Let $S = (T, V, H, t_{\text{fin}})$ be a restricted tiling system for which TilingWinner(S) is EXPTIME-hard and $w \in T^*$ an initial row. By definition of restricted tiling systems, the following holds for every tree s representing a winning strategy of Constructor.

(i) Each path represents a solution for S (with initial row w), and

(ii) Each node corresponding to a tile chosen by CONSTRUCTOR has exactly two children labelled by different tiles.

Thus, the following two statements are equivalent.

(a) $w \in \text{TilingWinner}(S)$.
(b) There is a strategy tree s for CONSTRUCTOR with the properties (i) and (ii).

In the following we define an encoding function enc that maps strategy trees fulfilling property (ii) to trees over alphabet $\{a, b, c, c'\}$. Furthermore, we construct from S and w a DFA-based XSD (B, λ), and a path query P such that the following are equivalent.

(c) There is a tree s' of the form $s' = \text{enc}(s)$ for some strategy tree s for CONSTRUCTOR with properties (i) and (ii).
(d) P is *not* valid w.r.t. (B, λ).

By combining the two above equivalences with the obvious equivalence between (b) and (c) we get that $w \in \text{TilingWinner}(S)$ if and only if P is *not* valid w.r.t. (B, λ). The theorem then follows because we have a reduction from the complement of TilingWinner(S) to the validity problem and the former is EXPTIME-complete because EXPTIME is closed under complementation.

The encoding of strategy trees is similar to the encoding of strings in the proof of Theorem 9. We basically replace nodes of the tree by paths of length $2k$, where $k = |T|$.

Let s be a strategy tree. We describe how the encoded tree enc(s) is obtained from s. We use the definitions of e_{ij} and enc(t_i) from Section 4.

For technical reasons that will become apparent below, we use the alphabet $\Sigma' = \{a, b, c, c'\}$ and allow additional encodings of tiles as follows. For every $i, j \in \{1, \ldots, k\}$ with $i < j$ we let

$$\text{enc}_i(t_j) \stackrel{\text{def}}{=} cc \cdots cc'c \cdots cbc \cdots ce_{i1} \cdots e_{ik},$$

be the string obtained from enc(t_j) by replacing the symbol c at position i by c'. In the following, we identify strings enc(t_i) and enc$_i(t_j)$ with paths consisting of $2k$ nodes that are labelled according to enc(t_i) and enc$_i(t_j)$, respectively.

We associate with each strategy tree s for CONSTRUCTOR an *encoded tree* enc(s) in two stages. The first stage proceeds in a top-down fashion. We replace the root with tile t by the path enc(t). We replace every node u that is the only child of its parent and is labelled with some t_i by enc(t_i). For all siblings u and v in s labelled by tiles t_i and t_j, respectively, with $i < j$, we replace u by enc(t_i) and v by enc$_i(t_j)$.

In the second stage we combine the two paths enc(t_i) and enc$_i(t_j)$ of a pair of siblings u, v by a *prefix tree* that is obtained by identifying their prefixes of length $i-1$ and put the resulting tree (or forest of two paths, if $i = 1$) below the lowest node of the encoding of the parent of u and v. After this, we no longer have siblings that carry the same label. The resulting tree is enc(s). We illustrate the encoding with an example.

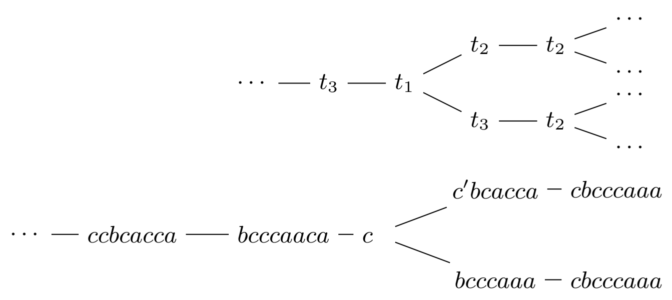

Fig. 2. At the top we see part of a strategy tree for Constructor in a game with four tile types. Below is the encoding of the same part of the tree.

Example 15 Figure 2 shows an example of how the encoding in the proof of Theorem 11 works. The example is meant to illustrate the idea of the encoding and does not represent a restricted tiling system. Assume that we have $T = \{t_1, t_2, t_3, t_4\}$, $\{(t_1, t_2), (t_1, t_3), (t_2, t_2), (t_3, t_1), (t_3, t_2)\} \subseteq H$, and $V = \{(t_1, t_3), (t_2, t_1), (t_3, t_2), (t_3, t_3), (t_4, t_1)\}$. On the top, we see a possible part of a strategy tree for Constructor, where the tile t_1 corresponds to a move of Constructor and t_2 and t_3 are the two possible next legal moves of Spoiler.

So, for example, $\text{enc}(t_1) = bcccaaca$, $\text{enc}(t_2) = cbcccaaa$, $\text{enc}(t_3) = ccbcacca$, and $\text{enc}(t_4) = cccbcaaa$. In Figure 2 we show part of a strategy tree for Constructor in a game. We use strings to represent unary tree fragments to simplify the picture. Groups of eight letters encode one tile (plus its vertical constraints). The encoding of the siblings t_2 and t_3 share a node because they have the same prefix c.

The DFA-based XSD (B, λ) is constructed from S and w as follows. The DFA B is a slight extension of the DFA $A'(S, w)$ constructed in the proof of Theorem 9. It tests, for a path in the given tree whether its label sequence is an encoding of a string $x \in T^*$ such that

- x has prefix w;
- the length of x is a multiple of $n \overset{\text{def}}{=} |w|$;
- the tiling function $\tau : \{1, \ldots, n\} \times \{1, \ldots, m\} \to T$ corresponding to x fulfills the horizontal constraints; and
- $\tau(n, m) = t_{\text{fin}}$.

Here, the encoding is over Σ' and allows substrings of the form $\text{enc}_i(t_j)$ beyond $\text{enc}(t_i)$ in even columns (chosen by Spoiler). We recall that the state set Q of $A'(S, w)$ basically[11] has states of the form (t, t', i), where t is the previous tile, t' the current tile and i a counter modulo $2kn$, and a rejecting sink state. In B, t' can also take the value "?" if the current tile is not yet determined but the DFA has already seen a c' in the encoding of the current tile (and thus the prefix

[11] As mentioned before, $A'(S, w)$ has further states for the prefix w.

tree has already branched). For the sake of clarity later in the proof, we briefly assume that B has accepting states (that, by definition, can only be reached after reading t_{fin}). These states are only important to define below where λ allows a node to be a leaf. We do not require accepting states in B in our DFA-based XSD.

The function λ is defined as follows.

– For states of the form $q = (t, \#, i)$ with $(i \mod 2k) < k$, for which $(t, t_{i \mod 2k}) \notin H$, $\lambda(q) = c$.
– For states of the form $q = (t, \#, i)$ with $(i \mod 2k) < k$, for which i indicates an odd column (where Constructor is about to move) and $(t, t_{i \mod 2k}) \in H$, $\lambda(q) = c + b$.
– For states of the form $q = (t, \#, i)$ with $(i \mod 2k) < k$, for which i indicates an even column (where Spoiler is about to move) and $(t, t_{i \mod 2k}) \in H$, $\lambda(q) = c + bc'$.
– For states of the form $q = (t, t', i)$ with $(i \mod 2k) < k$ and $t' \in T$, $\lambda(q) = c$.
– For states of the form $q = (t, ?, i)$ with $(i \mod 2k) < k$, $\lambda(q) = c + b$.
– For states of the form $q = (t, t', i)$ with $(i \mod 2k) \geq k$ and $t' \in T$, $\lambda(q) = e_{\ell j}$ where $t_\ell = t'$ and $j = i \mod 2k$.
– For states q corresponding to the first row, $\lambda(q)$ is just the next symbol from the encoding of w.
– For every "accepting state" q of B, $\lambda(q) = \varepsilon + b + c$.

Finally, the path query P has the form

$$\underbrace{a / * / * / \cdots / * / * / b}_{2nk-k+1 \text{ labels}}.$$

To complete the proof it only remains to show that (c) and (d) are indeed equivalent.

To this end, let us first assume that (c) holds, that is, there is a tree s' of the form $s' = \text{enc}(s)$ for some strategy tree s for Constructor with properties (i) and (ii). Since s is a strategy tree, each of its paths represents a solution for S with initial row w. As such, each path from root to leaf satisfies the horizontal and vertical constraints. Since it has the correct length and satisfies the horizontal constraints, B has a run over each path that ends in a state q such that $\lambda(q) = \varepsilon + b + c$. Since each node that corresponds to a Spoiler tile has exactly two children labelled by different tiles, we also have that, by construction, the conditions on λ are fulfilled. Thus, $\text{enc}(s)$ is satisfied by (B, λ). Finally, since s does not have any violations against the vertical constraints, we also have that P does not match s' by construction. Therefore s' is a witness for the fact that P is *not* valid w.r.t. (B, λ).

For the other direction, let us assume that P is *not* valid w.r.t. (B, λ). Let s' be a tree that conforms to (B, λ) and in which P does not match. By construction $s' = \text{enc}(s)$, for some tree s such that (ii) holds. Furthermore, as s' conforms to (B, λ) it follows that every path fulfills the horizontal constraints of S and ends with the final tile t_{fin}. Finally, as P does not match in s' there is no violation of any vertical constraint in s. Therefore, s also fulfills (i) and thus (c) holds. $\square$

In the appendix, we further show how Theorems 10 and 12 can be obtained by adapting the above proof.

References

1. M. Arenas, J. Daenen, F. Neven, J. Van den Bussche, M. Ugarte, and S. Vansummeren. Discovering XSD keys from XML data. In *SIGMOD*, 2013. To appear.
2. M. Benedikt, P. Bourhis, and P. Senellart. Monadic datalog containment. In *ICALP*, p.79–91, 2012.
3. M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2007.
4. H. Björklund, W. Gelade, and W. Martens. Incremental XPath evaluation. *ACM Trans. Database Syst.*, 35(4):29, 2010.
5. H. Björklund, W. Martens, and T. Schwentick. Optimizing conjunctive queries over trees using schema information. In *MFCS*, p.132–143, 2008.
6. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language XML 1.0 (fifth edition). World Wide Web Consortium, 2008.
7. B. S. Chlebus. Domino-tiling games. *JCSS*, 32(3):374–392, 1986.
8. J. Clark and S. DeRose. XML Path Language (XPath) version 1.0. Technical report, World Wide Web Consortium, 1999. http://www.w3.org/TR/xpath/.
9. J. Clark and M. Murata. Relax NG specification. http://www.relaxng.org, 2001.
10. S. Flesca, F. Furfaro, and E. Masciari. On the minimization of XPath queries. *J. ACM*, 55(1), 2008.
11. W. Gelade, T. Idziaszek, W. Martens, and F. Neven. Simplifying XML Schema: Single-type approximations of regular tree languages. In *PODS*, 2010.
12. K. Hashimoto, Y. Kusunoki, Y. Ishihara, and T. Fujiwara. Validity of positive XPath queries with wildcard in the presence of DTDs. In *DBPL*, 2011.
13. P. Kilpeläinen. Checking determinism of XML Schema content models in optimal time. *Inf. Syst.*, 36(3):596–617, 2011.
14. W. Martens, F. Neven, and T. Schwentick. Simple off the shelf abstractions for XML Schema. *SIGMOD Record*, 36(3):15–22, 2007.
15. W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML Schema. *ACM Trans. Database Syst.*, 31(3):770–813, 2006.
16. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.
17. F. Murlak, M. Oginski, and M. Przybylko. Between tree patterns and conjunctive queries: Is there tractability beyond acyclicity? In *MFCS*, p.705–717, 2012.
18. F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *LMCS*, 2(3), 2006.
19. L. Stockmeyer and A. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.
20. H. S. Thompson, N. Mendelsohn, D. Beech, and M. Maloney. XML Schema Definition Language (XSD) 1.1. http://www.w3.org/TR/xmlschema11-1/.

Appendix

By Σ we always denote a finite alphabet. By $a, b, c, \ldots$ we always denote elements from Σ. We use regular expressions r of the form

$$r ::= \varepsilon \mid a \mid (r \cdot r) \mid (r + r) \mid (r)^*,$$

where ε denotes the empty string and a ranges over symbols in the alphabet Σ. Sometimes we also use the symbol $\cdot$ for regular expression concatenation to improve readability. We sometimes abbreviate by Σ the disjunction $a_1 + \cdots + a_n$ where $\Sigma = \{a_1, \ldots, a_n\}$. As usual, we write $L(r)$ for the language defined by regular expression r. We define the *size* of regular expression r to be its total number of occurrences of alphabet symbols and operators. (In other words, the size of a regular expression is the number of nodes of its parse tree.) For example, both expressions $((a \cdot a) \cdot a)$ and $(a \cdot (b + c))$ have size five.For readability of expressions, we usually omit some brackets and abbreviate $r_1 \cdot r_2$ to $r_1 r_2$.

A *non-deterministic finite automaton* (NFA) A is a tuple $(\Sigma, Q, \delta, I, F)$, such that Q is a finite set of states, $I \subseteq Q$ is the set of initial states, F is the set of accepting states, and δ is the transition function of the automaton, defined as $\delta : Q \times \Sigma \to 2^Q$, mapping each pair of a state and symbol to a set of states. A run ρ of A on some string $w = a_1 \cdots a_n$ is a sequence of states $q_0, \ldots, q_n$, such that $q_0 \in I$ and for each $i \in [1, n]$, $q_i \in \delta(q_{i-1}, a_i)$. Furthermore, when $q_n \in F$, we say that a run is *accepting*. We define $A(w)$ to be the set of all states q such that there exists a run of A on w in which the last state is q. The string language accepted by A is denoted by $L(A)$ and is defined as the set of strings w for which there exists an accepting run of A on w (or, alternatively, $A(w)$ contains a state from F). A non-deterministic finite automaton A is said to be *deterministic* (or A is a DFA) if I is a singleton and the transition function maps each state/symbol-pair to a *singleton* set.

Proposition 8. *There is a restricted tiling system S, for which TilingWinner(S) is EXPTIME-hard.*

Proof (sketch). Again, EXPTIME-hardness for a fixed system S can be achieved by starting the proof of Theorem 5.1 in [7] from some fixed ATM M for which $L(M)$ is EXPTIME-complete. Property (1) is already guaranteed by the proof in [7]. It can be further required that M has exactly two transitions for every state-letter pair, existential states of M only occur at odd positions, and universal states only occur at even positions.[12] By enforcing these conditions and duplicating some of the tiles, we can guarantee property (2). Finally, as long as M has exactly two transitions for every state, property (3) is already enforced in [7], as SPOILER only has a real choice in the game when he chooses between two transitions. As already indicated above, we simply duplicate the only possible tile for SPOILER in cases where he has no choice,. □

[12] These requirements are already mentioned in [7] and they can be easily enforced.

Theorem 9. *Validity of regular expressions w.r.t. a DFA is PSPACE-complete even for regular expressions of the form $\Sigma^* a \Sigma^n b \Sigma^*$ and DFAs over the alphabet $\Sigma = \{a, b, c\}$.*

Proof. Obviously, the problem is in PSPACE since it can be reduced in logarithmic space to the containment problem for NFAs, which is known to be PSPACE-complete [19]. We show the lower bound by reduction from Tiling(S), i.e., corridor tiling with a fixed tiling system. Let $S = (T, V, H, t_{\text{fin}})$ with $T = \{t_1, \ldots, t_k\}$ be a tiling system such that Tiling(S) is PSPACE-hard. Notice that S exists by Theorem 6.

We first associate with every tiling function $\tau : \{1, \ldots, n\} \times \{1, \ldots, m\} \to T$ a string w_τ by simply concatenating all tiles in row-major order. More precisely, w_τ is the string from T^* of length mn that carries at position $(j-1)n + i$ tile $\tau(i, j)$, for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$.

From S and an initial row w a DFA $A(S, w)$ can be defined that tests whether a word $v \in T^*$ has the following properties:

- v has prefix w;
- the length of v is a multiple of $n \overset{\text{def}}{=} |w|$;
- the tiling function $\tau : \{1, \ldots, n\} \times \{1, \ldots, m\} \to T$ corresponding to v fulfills the horizontal constraints; and
- $\tau(n, m) = t_{\text{fin}}$.

$\boxed{\cdots}$ To this end, $A(S, w)$ uses states of the form (t, i) where

- $t \in T \uplus \{\#\}$ either represents the previous tile or, if $t = \#$, the fact that the first row of τ is being read, and
- $i \in \{0, \ldots, n-1\}$ is the column number of the previous symbol (and $i = 0$ at the beginning of each row).[13]

The definition of the transition function, the initial state and the set of final states of $A(S, w)$ is straightforward. We note that $A(S, w)$ has $O(n \cdot |T|)$ states.

However, for the actual reduction we use a more elaborate encoding of tiles and define the DFA accordingly. We simultaneously encode tiles and their relevant vertical constraints as strings of length $2k$. For each $i, j \in \{1, \ldots, k\}$ we let e_{ij} be a symbol that encodes whether $(t_i, t_j) \in V$ as follows.

$$e_{ij} \overset{\text{def}}{=} \begin{cases} a & \text{if } (t_i, t_j) \notin V \\ c & \text{otherwise.} \end{cases}$$

Then, for each $i \in \{1, \ldots, k\}$, we encode tile t_i as the string

$$\text{enc}(t_i) \overset{\text{def}}{=} cc \cdots cbc \cdots ce_{i1} \cdots e_{ik}$$

of length $2k$ in which the entry labeled b is at position i. For a string $v \in T^*$, we write enc(v) for the symbol-wise encoding of v.

[13] $A(S, w)$ has further states used to check that the string starts with w but we do not name them explicitly.

$\boxed{\cdots}$ It is straightforward to construct from $A(S, w)$ an automaton $A'(S, w)$ that accepts all encodings $\text{enc}(v)$ of strings $v \in L(A(S, w))$. The number of states is still polynomial in $|T| + |w|$. Basically, $A'(S, w)$ has states of the form (t, t', i) where i is a counter modulo $2kn$ and t and t' are the previous and current tile, respectively. [14] Again, $t = \#$ in positions corresponding to the first column and furthermore $t' = \#$ when the current tile has not yet been determined (because the b-position has not yet occurred).

$\boxed{\cdots}$ We claim that S has *no* solution with initial row w if and only if q_w matches all strings that are accepted by $A'(S, w)$.

For the "if"-claim, let us assume towards a contradiction that q_w matches all strings that are accepted by $A'(S, w)$ and there is a solution τ of S with initial row w. By construction, $\text{enc}(w_\tau)$ is accepted by $A'(S, w)$ and thus, by assumption, $\text{enc}(w_\tau)$ matches q_w. Let (i, j) be the position of the tile which is matched by the a of q_w (in the first match of q_w in $\text{enc}(w_\tau)$). Thus, the singleton a-symbol matches in the second half of the encoding $\text{enc}(\tau(i, j))$. As the intermediate $(a+b+c)$-block has length $(2n-1)k-1$, the singleton b-symbol of q_w matches in the first half of $\text{enc}(\tau(i, j+1))$. However, by definition of enc, this just means that $(\tau(i, j), \tau(i, j+1)) \notin V$, contradicting our assumption that τ is a solution. Thus, the "if"-claim holds.

For the "only if"-claim, let us assume that S has no solution with initial row w and that $u \overset{\text{def}}{=} \text{enc}(v)$ is accepted by $A'(S, w)$. By construction of $A(S, w)$ and $A'(S, w)$, v encodes a tiling function τ that has w as initial row, respects all horizontal constraints and has $\tau(n, m) = t_{\text{fin}}$. As S has no solution, τ thus needs to violate some vertical constraint. Let i and j be such that $(\tau(i, j), \tau(i, j+1)) \notin V$. It is not hard to see that, by definition of enc and construction of q_w, $\text{enc}(v)$ indeed matches q_w.

Altogether, we have established a reduction from the complement of Tiling(S) to Validity of regular expressions of the stated form w.r.t. a DFA. As PSPACE is closed under complementation this yields the theorem. □

Theorem 14. *Validity of tree pattern queries query w.r.t. an NTA is in EXPTIME.*

Proof (sketch). By Theorem 3.1 in [4], it is possible to compute in exponential time from the tree pattern query an exponential size NTA that accepts all trees that do *not* match the pattern.[15] By testing emptiness of the intersection of this NTA with the given NTA (in polynomial time in the size of the NTAs) we achieve the EXPTIME upper bound. □

Theorem 10. *Validity of tree pattern queries w.r.t. an NTA is EXPTIME-complete even for path queries of the form $a/*/*/\cdots/*/b$ over schemas with three symbols.*

[14] Again, $A'(S, w)$ has further states for the prefix w.

[15] This theorem actually shows that this is possible for much more general queries that correspond to Core XPath 1.0 queries and, in particular, allow negation.

Proof. Since a DFA-based XSD can be translated in polynomial time into an equivalent NTA we immediately have from Theorem 11 that Theorem 10 holds for schemas that use four symbols. However, in NTAs we can easily avoid the use of the symbol c'. More precisely, from the DFA-based XSD in the proof of Theorem 11 an NTA can be constructed in polynomial time that accepts the same tree language except that c' is relabelled to c. The rest of the proof is analogous and we therefore also obtain Theorem 10.

By a slightly different encoding of strategy trees we are now also able to give the proof for Theorem 12.

Theorem 12. *Validity of tree pattern queries w.r.t. a DTD is EXPTIME-complete even for tree pattern queries of the form* $*[/a]/*/\cdots/*/b$ *over DTDs.*

Proof. This proof follows a similar argument as the proof of Theorem 11, but we need to use a slightly different encoding. In particular, our alphabet is no longer constant.

The DTD D accepts trees which are obtained from the language of the DFA-based XSD in the proof of Theorem 11 as follows. Let s' be a tree that is in the language of (B, λ). For each node u of s', we replace its label $x \in \{a, b, c, c'\}$ by the pair (x, q), where $q = B(\text{anc-str}^{s'}(u))$. It is well-known that this language can be defined by a DTD.[16] Let D' be such a DTD.

We now do one more change to the tree language: Each node that is labelled (a, q) (resp. (b, q)) for some state q of B, has an additional child labelled a (resp., b). This newly obtained language can also easily be defined by a DTD: If $D' = (\Sigma, d', S)$ with $d'((a, q)) = R_{a,q}$ then $D = (\Sigma, d, S)$ with $d((a, q)) = R_{a,q} \cdot a$ (similarly, $d((b, q)) = R_{b,q} \cdot b$ for $R_{b,q} = d'((b, q))$). Furthermore, we define $d(a) = d(b) = \varepsilon$. As such, in $L(D)$ we have that s' has a b-labelled node $2nk - k$ levels below an a-labelled node if and only if the newly obtained tree has a b-labelled node that is $2nk - k$ levels below a node that has an a-labelled child.

The query Q is then of the form

$$\underbrace{*[/a]/*/*/\cdots/*/*/b}_{2nk-k+2 \text{ labels}}.$$

This query has one more label than the path query P in the proof of Theorem 11 because its root node has an extra child (labelled a).

Due to the same argument as in the proof of Theorem 11, we have that q is valid w.r.t. D if and only if Constructor does not have a winning strategy in the corridor tiling game. $\square$

[16] This is due to the definition of extended DTDs (EDTDs), which are equivalent to NTAs.