Consistency of injective tree patterns

Citation for published version:
David, C, Francis, N & Murlak, F 2014, Consistency of injective tree patterns. in V Raman & SP Suresh (eds), *Proceedings of the 34th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'14).* vol. 29, Leibniz-Zentrum für Informatik, New~Dehli, India, pp. 279-290. https://doi.org/10.4230/LIPIcs.FSTTCS.2014.279

Digital Object Identifier (DOI):
10.4230/LIPIcs.FSTTCS.2014.279

Published In:
Proceedings of the 34th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'14)

Consistency of Injective Tree Patterns

Claire David[1], Nadime Francis[2], and Filip Murlak[3]

1 **Université Paris-Est Marne,** `Claire.David@univ-mlv.fr`
2 **ENS Cachan,** `francis@lsv.ens-cachan.fr`
3 **University of Warsaw,** `fmurlak@mimuw.edu.pl`

Abstract

Testing if an incomplete description of an XML document is consistent, that is, if it describes a real document conforming to the imposed schema, amounts to deciding if a given tree pattern can be matched injectively into a tree accepted by a fixed automaton. This problem can be solved in polynomial time for patterns that use the child relation and the sibling order, but do not use the descendant relation. For general patterns the problem is in NP, but no lower bound has been known so far. We show that the problem is NP-complete already for patterns using only child and descendant relations. The source of hardness turns out to be the interplay between these relations: for patterns using only descendant we give a polynomial algorithm. We also show that the algorithm can be adapted to patterns using descendant and following-sibling, but combining descendant and next-sibling leads to intractability.

1998 ACM Subject Classification H.2.1 [Database Management]: Logical Design – Data models, F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems

Keywords and phrases XML, incomplete information, injective tree patterns, consistency

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.279

1 Introduction

It is convenient to think that a database instance is a faithful representation of a fragment of reality; but, in fact, it almost never is. Pieces of information are not available, or classified, or get lost on the way due to storage and transmission failures. Additional sources of incompleteness are complex data management tasks, like data integration [11] or data exchange [6]. Since the seminal work of Imielinski and Lipski [8], incompleteness of information has been an important topic in relational database theory [7]. More recently, the need to deal with incomplete information has increased dramatically, due to large amounts of data on the Web [1]. This data tends to be more prone to errors than data stored in traditional DBMSs, and transformation, integration, and exchange of data between different applications is inherent to this context. Dealing with data on the Web also means facing new data models such as XML documents or graph databases, and scenarios involving incomplete information for such models have been considered [4, 9].

Incompleteness brings new difficulties into classical tasks such as query answering (what does it mean to answer a query over an incomplete database?), but it also gives rise to new tasks. One of such problems is consistency: is there a real instance that matches the incomplete description? A systematic study of problems related to incomplete XML data was undertaken in [2]. XML documents are modelled as unranked labelled trees. For such a tree there are several kinds of information that can be missing in the description: nodes can be missing, or their labels, or their relative position in the tree. Thus an incomplete tree can be seen as a tree with some labels missing, and some edges representing descendant relation, rather than child relation (one can also allow partial information about sibling order).

Assuming the so-called DOM semantics, nodes of XML documents have their identity, which is never lost (if it gets lost, the node is considered to be lost). On its own, such description is always consistent: we obtain a proper document by turning all edges to child edges and filling in the labels arbitrarily. Typically, however, the setting also involves a schema (DTD, XSM, RelaxNG), that describes the shape of correct documents. The structural restrictions of the schema can always be expressed by a tree automaton. Thus, the consistency problem for a fixed schema amounts to deciding if there is a tree accepted by the automaton, that matches the given incomplete description.

The incomplete descriptions of [2] coincide with the notion of tree patterns, originally introduced as an elegant formalism to express acyclic conjunctive queries over trees and extensively studied in connection with the XPath query language [3, 12, 13, 14, 15]. Our consistency problem is a variant of the satisfiability problem for tree patterns with respect to a fixed automaton [3]. The difference lies in the semantics. Classically, a pattern is satisfied in a tree if its nodes can be mapped to the tree nodes in such a way that the labels and relations are preserved. In our setting, the DOM semantics imposes an additional requirement: the mapping has to be injective. This makes the existing results on patterns inapplicable. We also cannot use the variant of the injective semantics considered in [5], where it is additionally assumed that if two pattern nodes are incomparable (neither is descendant of the other), they must be mapped to incomparable nodes in the tree.

Already in [2] it is noticed that the consistency problem is in NP, but the exact complexity is left open. For a special case of patterns (incomplete descriptions) that do not involve descendant edges, a polynomial algorithm is given. In a highly nontrivial extension of this result, Kopczynski [10] gives a polynomial procedure for patterns that contain at most one descendant edge on each branch.[1]

In this paper we close the gap: we show that the consistency problem is NP-complete. In fact our result is tight with respect to Kopczynski's polynomial algorithm: the problem is NP-hard already for patterns with at most two descendant edges per branch. We also investigate further the sources of hardness and find out that for descendant-only patterns the problem can be solved in PTime. Finally, we consider possible extensions involving the sibling order. Combining next-sibling with descendant leads to intractability, but for patterns using only descendant and following-sibling an adaptation of our proof techniques gives tractability.

2 Preliminaries

For an unranked Σ-labelled tree T, we write $nodes_T$ for the set of nodes, $root_T$ for the root of T, and $lab_T(v)$ for the label of a node v in T. We also use the notation $u \downarrow v$ and $u \downarrow^+ v$ to indicate that node v is, respectively, a child or a descendant of node u. We write T_v for the subtree of tree T rooted at node v.

An *antichain* in a tree is any sequence of nodes such that no two of them are in the descendant relation (they can be siblings). A *frontier* is a maximal antichain that does not contain the root of the tree.

▶ **Definition 1.** A *tree pattern* π over the alphabet Σ is a finite unranked Σ-labelled tree, whose edges are of one of two kinds: child edges, denoted $\downarrow$, and descendant edges, denoted

[1] In fact, Kopczynski gives an algorithm for the general problem, but under his own semantics, resembling that of [5]. For patterns with at most one descendant per branch, this semantics coincides with the standard injective semantics.

$\downarrow^+$. We write $lab_\pi(v)$ for the label of v in π. We also use notation $u \downarrow v$ and $u \downarrow^+ v$ to indicate that the nodes are connected with a $\downarrow$-edge or $\downarrow^+$-edge, respectively.

▶ **Definition 2.** A tree pattern π is *satisfied* in a tree T, written as $T \models \pi$, if there exists an *injective homomorphism* $h \colon \pi \to T$, that is, an injective function mapping the nodes of π to nodes of T that preserves the labels and the relations, that is, for all nodes u, v in π

- $lab_T(h(v)) = lab_\pi(v)$;
- if $u \downarrow v$ in π, then $h(u) \downarrow h(v)$ in T;
- if $u \downarrow^+ v$ in π, then $h(u) \downarrow^+ h(v)$ in T.

▶ **Definition 3.** A *tree automaton* $\mathcal{A} = (\Sigma, Q, \delta, F)$ consists of an alphabet Σ, a finite set of states Q, a set of final states $F \subseteq Q$, and a transition function $\delta \colon \Sigma \times Q \to \mathcal{P}(Q^*)$, assigning regular languages over Q (represented as regular expressions) to each label and state.

A *run* of $\mathcal{A}$ over a tree T is a labelling ρ of the nodes of T with elements of Q such that for each node of v, if v has children $v_1, v_2, \ldots, v_k$, then $\rho(v_1)\rho(v_2)\ldots\rho(v_k) \in \delta(lab_T(v), \rho(v))$. If v is a leaf, this amounts to $\varepsilon \in \delta(lab_T(v), \rho(v))$.

A run ρ is *accepting* if the root's label is in F. If T admits an accepting run, we say that T is accepted by $\mathcal{A}$. We write $L(\mathcal{A})$ for the language recognized by $\mathcal{A}$, i.e., the set of trees accepted by $\mathcal{A}$. A state q is *productive* if it occurs in some accepting run.

Let $\mathcal{A}$ be a tree automaton. We are interested in the complexity of the following problem.

PROBLEM:	$\textsc{Cons}_\mathcal{A}$
INPUT:	Tree pattern π.
QUESTION:	Is there a tree $T \in L(\mathcal{A})$ such that $T \models \pi$?

Note that the automaton $\mathcal{A}$ is not part of the input. The complexity is measured in terms of the size of the pattern π. In the context of the incomplete information scenario, where π represents information about an XML document, this corresponds to *data complexity* of consistency.

3 NP-hardness

We first consider the problem for patterns with full vertical navigation, that is, with $\downarrow$ and $\downarrow^+$ edges.

▶ **Theorem 4.** *There is an automaton $\mathcal{A}$ such that $\textsc{Cons}_\mathcal{A}$ is* NP*-complete. Moreover, $\textsc{Cons}_\mathcal{A}$ is* NP*-hard already for patterns with at most two occurrences of $\downarrow^+$ per branch.*

Proof. The NP upper bound can be proved by a standard guess and check technique [2]. The rest of this proof is devoted to showing that the problem is NP-hard.

Consider the language K defined in Figure 1. It is straightforward to construct an automaton recognizing K. We claim that for any automaton $\mathcal{A}$ recognizing K, $\textsc{Cons}_\mathcal{A}$ is NP-hard (even for patterns with at most two occurrences of $\downarrow^+$ per branch).

We reduce from CNF-SAT. Let $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ be a conjunction of clauses over variables $x_1, x_2, \ldots, x_n$. We build a pattern π_φ such that the formula φ is satisfiable if and only if the pattern π_φ is satisfiable in a tree T from K.

The pattern π_φ can be decomposed in two parts. One part ensures that the tree T represents precisely the formula φ. The rest of the pattern represents a valuation of the variables $x_1, x_2, \ldots, x_n$ and the proof that this valuation satisfies the formula φ. The idea of the encoding of the formula into a tree T from K is to associate each variable x_i with an

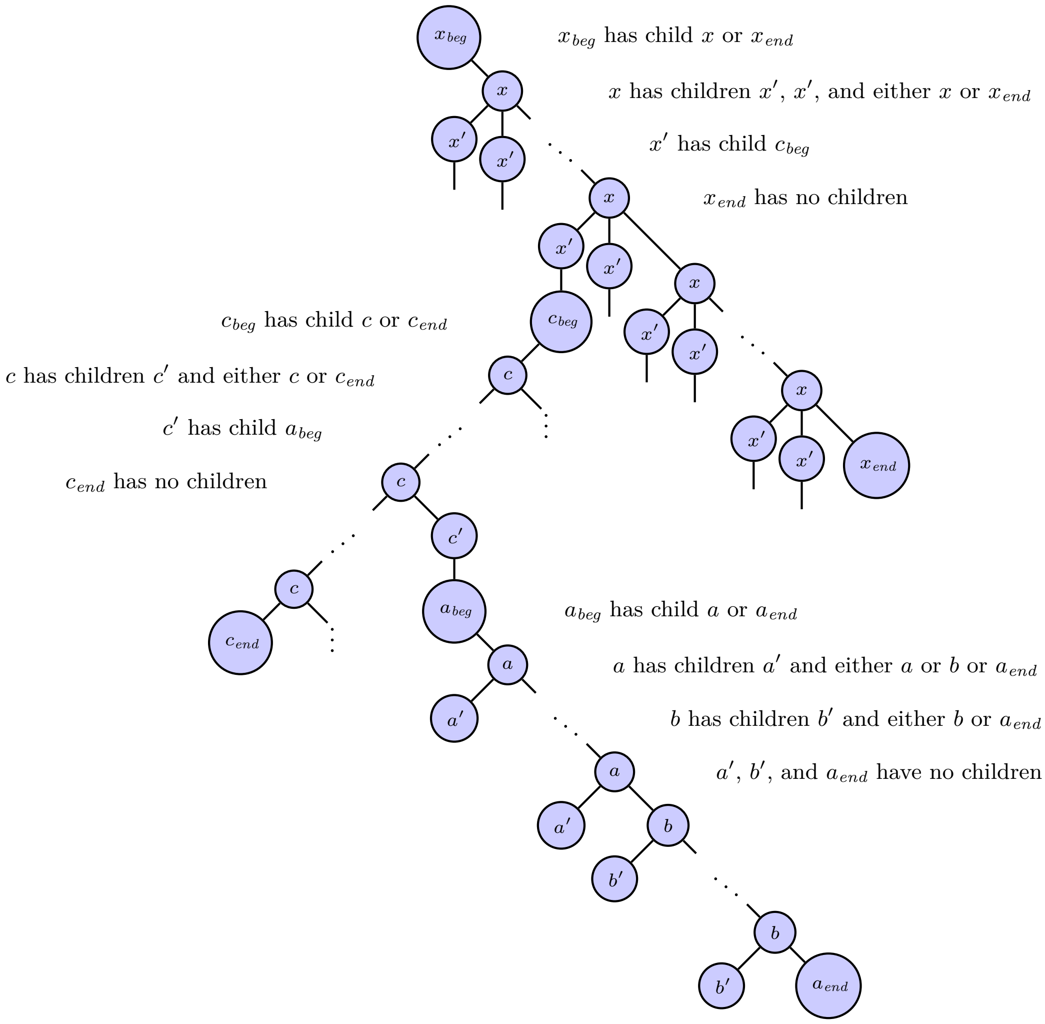

Figure 1 The tree language recognized by the automaton $\mathcal{A}$ used in the reduction of CNF-SAT to $\text{CONS}_{\mathcal{A}}$.

x node and encode in the two corresponding x'-rooted subtrees two lists of clauses: those satisfied when x_i is true, and those satisfied when it is false.

The full pattern π_φ is given in Figure 2. Notice that subpattern F^i_j depends on whether literal $\bar{x}_i$ occurs in the clause c_j or not; subpattern T^i_j is defined analogously, with literal $\bar{x}_i$ replaced with x_i. Let π'_φ be the pattern obtained from π_φ by removing subpatterns V^i and C^j for all i and j. In other words, we keep the blue nodes, but remove the green and red nodes. Observe that whenever π'_φ is matched in a tree $T \in K$, the subpatterns T^i and F^i must be matched at the grandchildren of the ith x node. Indeed, for T^n and F^n there is no choice. Consequently, since the matching must be injective, for T^{n-1} and F^{n-1} there is no choice either, etc. A similar argument applies to the subpatterns T^i_j and F^i_j. This implies that (up to the ordering of x' siblings) there is exactly one tree in K satisfying π'_φ: the tree T_φ obtained from π'_φ by filling in the missing nodes with labels x', c', a', b'. Moreover, there is exactly one injective homomorphism from π'_φ to T_φ, that is the one induced by the construction of T_φ.

Intuitively, the subpattern T^i lists the clauses of φ that are made true by setting x_i to

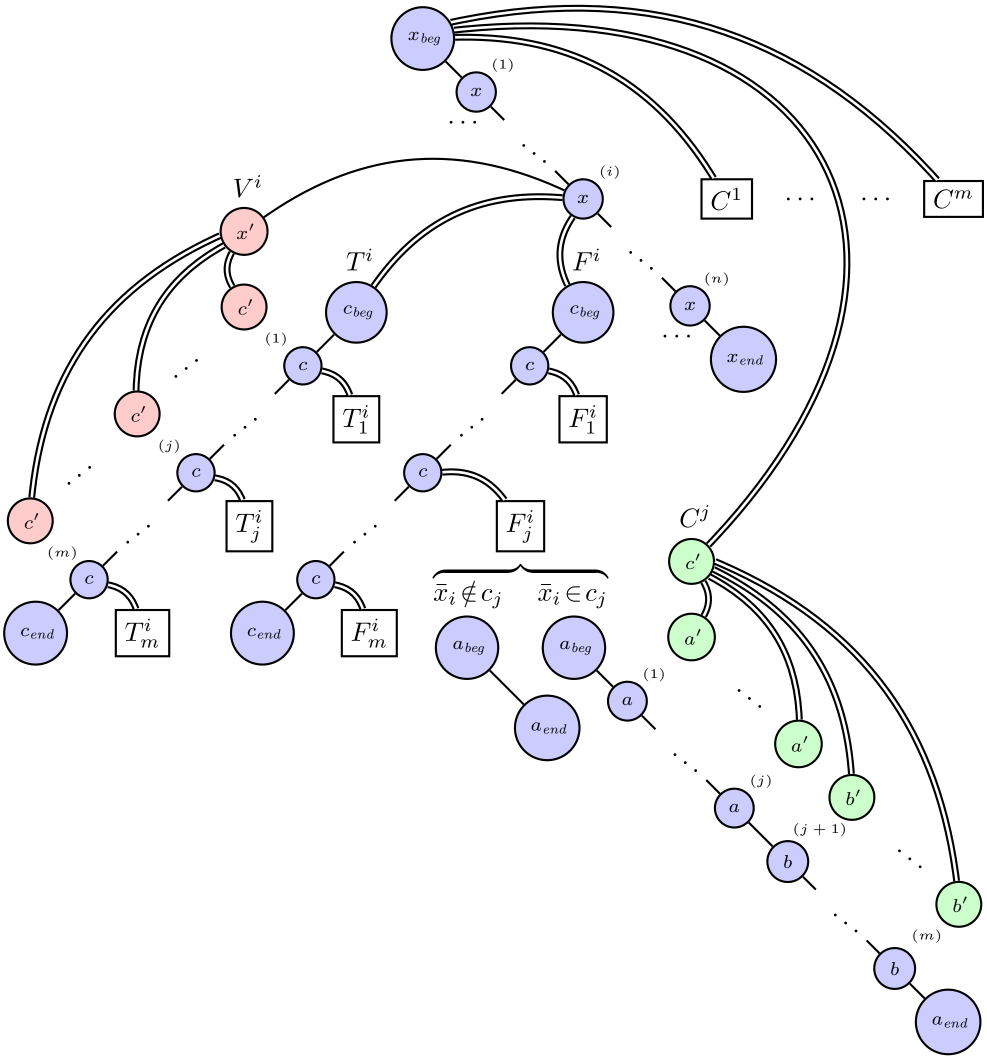

Figure 2 The pattern encoding a CNF formula $c_1 \wedge c_2 \wedge \cdots \wedge c_m$ over variables $x_1, x_2, \ldots, x_n$. Single and double lines represent child and descendant edges, respectively.

true, and F^i lists the ones made true by setting x_i to false. Whether clause c_j is true or not is encoded by subpatterns T^i_j and F^i_j: a sequence of j labels a and $m-j$ labels b is inserted between a_{beg} and a_{end} if and only if clause c_j is made true.

It remains to show that this homomorphism can be extended to the full pattern π_φ if and only if φ is satisfiable. There are two ways of matching V^i in T_φ: at the parent of the image of T^i or at the parent of the image of F^i. In either case, the matching uses all c' nodes in the corresponding subtree, while the nodes in the other subtree remain unused. Thus, choosing T^i should be interpreted as setting x_i to false, since c' nodes under F^i remain unused, and choosing F^i as setting x_i to true, since c' nodes under T_i remain unused. When all subpatterns V^i have been matched, subpattern C^j can be matched if and only if the associated valuation makes clause c_j true.

It follows that $T_\varphi \models \pi_\varphi$ if and only if there exists a valuation of the variables $x_1, x_2, \ldots, x_n$ that makes true every clause of φ. ◀

4 Descendant-only patterns

In the previous section we have proved that $\textsc{Cons}_{\mathcal{A}}$ is NP-complete in general. We know that the problem is tractable for some restricted classes of patterns such as patterns using only child relation [2] or the class considered by Kopczynski [10]. In this section, we prove that $\textsc{Cons}_{\mathcal{A}}$ is also tractable for tree patterns that only use the descendant relation.

▶ **Theorem 5.** *Let $\mathcal{A}$ be a fixed tree automaton. Then $\textsc{Cons}_{\mathcal{A}}$ is solvable in* PTime *for $\downarrow^+$-only tree patterns.*

The key argument to prove Theorem 5 is that consistency of a descendant-only tree pattern with respect to an automaton $\mathcal{A}$ can be reduced to membership of the underlying tree of the pattern in a regular tree language that depends only on $\mathcal{A}$. When the automaton $\mathcal{A}$ is fixed, the latter can be checked in time polynomial in the size of π. This stronger result is proved in Lemma 14. The remaining of the section is dedicated to a fine analysis of descendant-only tree patterns together with a tree automaton, providing the tools needed to state and prove this lemma.

Our goal is to build concise representations of trees in $L(\mathcal{A})$ that satisfy some descendant-only pattern π, in such a way that the size of these representations does not depend on π. The first step is to omit nodes that are not used to satisfy π. The notion of *descendant count* introduced in Definition 6 provides a concise way to represent the set of possible frontiers that are reachable starting from a given label-state pair in a run of $\mathcal{A}$.

▶ **Definition 6.** Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. A *count* for $\mathcal{A}$ is a function $\alpha : \Sigma \times Q \to \overline{\mathbb{N}}$, where $\overline{\mathbb{N}} = \mathbb{N} \cup \{*\}$, with the natural order extended with $i \leq *$ for all $i \in \mathbb{N}$. We say that count α is smaller than count β if $\alpha(a, q) \leq \beta(a, q)$ for all pairs $(a, q) \in \Sigma \times Q$.

We say that a count α is *realized* at (a, q) if for all $n \in \mathbb{N}$, there exists a tree T, a run ρ of $\mathcal{A}$ on T, and a frontier w in T such that

- the root v of T has label a and $\rho(v) = q$;
- for all $(a', q') \in \Sigma \times Q$ such that $\alpha(a', q') \in \mathbb{N}$, w contains at least $\alpha(a', q')$ nodes v with label a' and such that $\rho(v) = q'$;
- for all $(a', q') \in \Sigma \times Q$ such that $\alpha(a', q') = *$, w contains at least n nodes v with label a' and such that $\rho(v) = q'$.

Finally, given $(a, q) \in \Sigma \times Q$, the *descendant count* of a and q, denoted by $\mathrm{DC}_{\mathcal{A}}(a, q)$, is defined as the set of all maximal counts for $\mathcal{A}$ that are realized at (a, q).

▶ Remark. The sets $\mathrm{DC}_{\mathcal{A}}(a, q)$ are finite and can be computed. Indeed, we can easily compute a context-free grammar recognizing the set $\mathrm{Fr}_{\mathcal{A}}(a, q) \subseteq (\Sigma \times Q)^*$ of sequences of letter-state pairs yielded by the frontiers occurring in the definition of $\mathrm{DC}_{\mathcal{A}}(a, q)$. As $\mathrm{Fr}_{\mathcal{A}}(q, a)$ is closed under subsequences, its Parikh image is a (finite) union of linear sets of the form $\{\beta \in \mathbb{N}^{\Sigma \times Q} \mid \beta \leq \alpha\}$, where α is a count. Since a semilinear representation of the Parikh image of a context-free language can be computed effectively, the involved counts α can be deduced as well. $\mathrm{DC}_{\mathcal{A}}(a, q)$ consists of the maximal ones among them.

Using descendant counts, we define the notion of skeleton for a tree automaton $\mathcal{A}$ which can be seen as a sparse representation of a tree in $L(\mathcal{A})$, where some nodes are omitted. We show that if a tree pattern π is satisfied by a skeleton s for $\mathcal{A}$, then it is consistent with $\mathcal{A}$.

▶ **Definition 7.** Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. A *skeleton* s for $\mathcal{A}$ is a tree whose nodes carry a label from $\Sigma \times Q$ and can optionally be flagged as starred. Additionally, for each node v of s with label (a, q), there exists $\alpha \in \mathrm{DC}_{\mathcal{A}}(a, q)$ such that for all (a', q')

- if $\alpha(a',q') \in \mathbb{N}$, then v has at most $\alpha(a',q')$ children with label (a',q'), all non-starred;
- if $\alpha(a',q') = *$, then v has an arbitrary number of children with label (a',q'), all starred;
- if v is the root, then v is not starred, and q is productive.

We say that s *satisfies* a $\downarrow^+$-only tree pattern π if the underlying tree of s satisfies π.

Descendant counts are used to build skeletons and ensure that each level of the skeleton is consistent with $\mathcal{A}$ and can indeed be simulated by a tree in $L(\mathcal{A})$. This is more precisely shown in the following lemma, where we prove that, starting from a skeleton s, we can build a tree T in $L(\mathcal{A})$ that features the same nodes as s, arranged in the same descendant order.

▶ **Lemma 8.** *Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton and s be a skeleton for $\mathcal{A}$. Then there exists a tree T, a run ρ of $\mathcal{A}$ on T and an injective mapping $i : nodes_s \to nodes_T$ such that, for all nodes u, v of s,*
- *if $lab_s(u) = (a,q)$, then $lab_T(i(u)) = a$ and $\rho(i(u)) = q$;*
- *if $u \downarrow v$ in s, then $i(u) \downarrow^+ i(v)$ in T;*
- *if u is the root of s, then $i(u)$ is the root of T.*

Proof. We prove this by induction on the structure of s.

Assume that s consists of a single node u with label (a,q). By Definition 7, there exists a count $\alpha \in \mathrm{DC}_{\mathcal{A}}(a,q)$. Since α is realized at (a,q), the tree T of Definition 6 satisfies the requirements of the lemma.

Assume that u is the root of s, with children $s_1, \ldots, s_n$. Let (a,q) be the label of u, and (a_i, q_i) be the label of the root of s_i. Then, by definition of s, there exists a count α that is realized at (a,q) and fits the definition of s at u. Then, by Definition 6, there exists a tree T and a run ρ on T such that T has root v with $lab_T(v) = a$, $\rho(v) = q$, and with some frontier $v_1 \ldots v_n$ with $lab_T(v_i) = a_i$ and $\rho(v_i) = q_i$. Then we can build from T the required tree by replacing the nodes of this frontier with the trees $T_1, \ldots, T_n$ produced by the induction hypothesis applied to $s_1, \ldots, s_n$. ◀

Since the root of a skeleton is always labeled by a productive state and our patterns only use $\downarrow^+$, Lemma 8 implies the following result.

▶ **Corollary 9.** *Let $\mathcal{A}$ be an automaton, s a skeleton for $\mathcal{A}$ and π a $\downarrow^+$-only pattern. If a skeleton s satisfies π, then there exists a tree $T \in L(\mathcal{A})$ that satisfies π.*

Note that, even though skeletons can be sparser than trees, there is still an infinite number of them. We show that we can represent all skeletons considering only the finite set of *reduced skeletons*.

▶ **Definition 10.** Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. A *reduced skeleton* s for $\mathcal{A}$ is a skeleton that additionally satisfies the following two properties:
- each pair label-flag appears at most once in each branch of s;
- each node of s has at most one starred child of each label.

Note that the number of reduced skeletons is finite for any automaton $\mathcal{A}$. Indeed, reduced skeletons are both bounded in depth, as there are a finite number of labels and they are not allowed to repeat along a branch, and in width, since the maximum number of non-starred children of any given label is bounded by the largest value different from $*$ taken by any of the counts in $\bigcup_{(a,q)\in\Sigma\times Q} \mathrm{DC}_{\mathcal{A}}(a,q)$.

Intuitively these skeletons correspond to minimal ones and can be obtained by pruning long branches and large siblings sets in some larger skeleton. Moreover, reduced skeletons contain enough information to recover the whole skeletons, by means of the horizontal and vertical pumping properties of tree automata.

▶ **Definition 11.** Skeleton s reduces to skeleton s' if s' can be obtained from s by applying a (possibly empty) sequence of the following reductions:

(H) Remove any starred node of s that has the same label as some of its starred siblings.
(V) Assume that a node u of s and its descendant v carry the same labels and flags. Then reduce s to a skeleton obtained by replacing in s the subtree s_u with s_v.

We write $\mathrm{red}(s)$ for the set of skeletons to which s reduces, that cannot be further reduced.

Note that the label and flag of the root of s are preserved by both reduction steps. Also, if s reduces to s', and s is a skeleton for $\mathcal{A}$ then s' is also a skeleton for $\mathcal{A}$. Moreover if s cannot be reduced by either (H) or (V), then s is a reduced skeleton. This implies that $\mathrm{red}(s)$ is the set of all reduced skeletons s' such that s reduces to s'.

The reductions (H) and (V) give a way to simplify a skeleton. The final ingredient we need is a way of combining skeletons without losing information. To this end we define the notion of *injection* of a skeleton into another. Intuitively an injection of s_2 into s_1 can be viewed as a skeleton s expanding s_1 such that s_2 can be matched disjointly from s_1 into s.

▶ **Definition 12.** Let s, s_1 and s_2 be skeletons. Then s is an *injection* of s_2 into s_1 if there exists two injective mappings $i_1 : nodes_{s_1} \to nodes_s$ and $i_2 : nodes_{s_2} \to nodes_s$ such that

- if u is the root of s_1, then $i_1(u)$ is the root of s;
- the images of i_1 and i_2 are disjoint;
- mappings i_1 and i_2 preserve labels and flags as well as descendant relation.

▶ Remark. Note that if s_1 satisfies a pattern π_1 and s_2 satisfies a pattern π_2, then any injection of s_2 into s_1 satisfies π_1 and π_2 *simultaneously*, that is, we can match π_1 and π_2 in such a way that their images are disjoint.

We are now ready to define the tree automaton $\mathcal{A}_\Pi$ and prove that it recognizes the set of all descendant-only tree patterns that are consistent with a given tree automaton $\mathcal{A}$. As explained in the beginning of the section Theorem 5 follows directly from this result.

▶ **Definition 13.** Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. Then we define the *pattern automaton* $\mathcal{A}_\Pi = (\Sigma, Q_\Pi, \delta_\Pi, F_\Pi)$ of $\mathcal{A}$ as follows.

- $Q_\Pi = F_\Pi$ is the set of all reduced skeletons for $\mathcal{A}$.
- Let s be a reduced skeleton for $\mathcal{A}$ and $a \in \Sigma$, then $s_1 \dots s_n \in \delta(s, a)$ if and only if there exist skeletons $t_0, \dots, t_n$ such that
  - t_0 is the root of s and is labeled (a, q) for some q;
  - for all $i > 0$, there exists an injection of s_i into t_{i-1} that reduces to t_i;
  - $t_n = s$ (or $t_0 = s$ if $s_1 \dots s_n$ is ε.)

It is easy to check that $\mathcal{A}_\Pi$ is a properly defined tree automaton. Indeed, the three properties defining $\delta(s, a)$ actually define the initial states, transitions and final states of a finite automaton, hence $\delta(s, a)$ is regular.

▶ **Lemma 14.** *Let $\mathcal{A}$ be a tree automaton and π be a $\downarrow^+$-only tree pattern. Then π is consistent with respect to $\mathcal{A}$ if and only if $\pi \in L(\mathcal{A}_\Pi)$.*

Proof. $(\Rightarrow)$ Assume that π is consistent with respect to $\mathcal{A}$. We want to exhibit an accepting run ρ of $\mathcal{A}_\Pi$ on π.

Let $T \in L(\mathcal{A})$ such that $T \models \pi$, which means that there is an injective homomorphism h from π to T. Let μ be an accepting run of $\mathcal{A}$ on T. Combining, the tree T, the run μ and the pattern π, we build a skeleton s as follows:

- the nodes of s correspond to the nodes in $h(\pi)$;
- for each node v of π with label a, the corresponding node in s has label $(a, \mu(h(v)))$;
- the father of a node v in s is its closest ancestor in T that also belongs to $h(\pi)$;
- for each node v of s of label (a, q), choose $\alpha \in \mathrm{DC}_{\mathcal{A}}(a, q)$ such that the number of v's children of label (a', q') is at most $\alpha(a', q')$, and flag the children as starred accordingly.

Regardless of the choices of α, the resulting s is indeed a properly defined skeleton for $\mathcal{A}$, as T and μ witness all the required descendant counts. Note also that s satisfies π through the same injective homomorphism h.

For all nodes v of π, we define π_v as the subpattern of π rooted at v. For V, a subset of the set of nodes of π, we deduce s_V^0 from s by keeping only the least common ancestor of nodes in V as well as all the nodes of s that appear in $h(\pi_v)$ for all $v \in V$, and linking nodes to their closest ancestor, as it is done for s. For s_V^0 to be a proper skeleton, we also unflag its root in case it is flagged as starred. We also define s_V as any skeleton arbitrarily chosen in $\mathrm{red}(s_V^0)$. If V consists of a single node v, we simply write s_v^0 and s_v.

We are now ready to exhibit an accepting run ρ of $\mathcal{A}_\Pi$ on π. For each node v of π, we define $\rho(v) = s_v$. It remains to show that ρ is a properly defined run of $\mathcal{A}_\Pi$; it will immediately be accepting, as all states of $\mathcal{A}_\Pi$ are final. We show by induction on the structure of π that, for all nodes v of π, the partial run defined by ρ on π_v is a correct run for $\mathcal{A}_\Pi$.

Let v be a leaf node of π with label a. Then s_v^0 is a skeleton consisting of a single node labeled (a, q) for some q, and is thus reduced. Hence, $\rho(v) = s_v = s_v^0$, $\varepsilon \in \delta_\Pi(a, s_v)$ and ρ is a properly defined run on π_v.

Let v be an internal node of π with label a. Let $u_1, \ldots, u_n$ be the children of v. By the induction hypothesis, we know that ρ is a properly defined run on all π_{u_i}. Let t_0 be the root of s_v. As $h(v) = t_0$, then t_0 has label (a, q) for some q. For all $i > 0$, we define $V_i = \{u_1, \ldots, u_i\}$ and $t_i = s_{V_i}$. Then, this sequence of skeletons satisfies the definition of $\mathcal{A}_\Pi$. The injection of s_{u_i} into t_{i-1} is simply $s_{V_i}^0$, which reduces to t_i by definition. Hence, ρ is a properly defined run on π_v.

($\Leftarrow$) Assume that $\pi \in L(\mathcal{A}_\Pi)$. Let ρ be an accepting run of $\mathcal{A}_\Pi$ on π. For each node v of π, we define π_v as the subpattern of π rooted at v. We now prove by induction on the structure of π that, for all nodes v of π, there exists a skeleton s that satisfies π_v and that reduces to $\rho(v)$.

Let v be a leaf node of π with label a. By definition of $\mathcal{A}_\Pi$, the reduced skeleton $\rho(v)$ is a single node labeled (a, q) for some q. Then $\rho(v)$ satisfies π_v and is already reduced. Hence, we can choose $s = \rho(v)$.

Let v be an internal node of π labeled a. Assume that v has only two children, v_1 and v_2, as other cases are similar. Let u be the root of $\rho(v)$. By definition of $\mathcal{A}_\Pi$, there is an injection t of $\rho(v_1)$ and $\rho(v_2)$ into u that reduces to $\rho(v)$. By induction, there are two skeletons s_1 and s_2 that respectively reduce to $\rho(v_1)$ and $\rho(v_2)$ and respectively satisfy π_{v_1} and π_{v_2}.

We can build from t a skeleton s by reverting in t all the reductions steps that are used to reduce each s_i to $\rho(v_i)$, as well as adding enough copies of starred nodes of t so that s is an injection of s_1 and s_2 into u. Thus, s satisfies both π_{v_1} and π_{v_2} simultaneously without using the root node. Moreover, it is easy to check that s reduces to $\rho(v)$, since all new nodes can simply be removed by reductions steps. Let h be an injective homomorphism that witnesses the fact that s satisfies π_{v_1} and π_{v_2} simultaneously and without using the root node. Then we can extend h by mapping v to u. This extended mapping witnesses the fact that s satisfies π_v, as u has label (a, q) for some q, since it is the root of $\rho(v)$.

By applying this induction to the root v of π, we deduce that there exists a skeleton s that reduces to $\rho(v)$ and satisfies π. We conclude using Lemma 8 and Corollary 9. ◀

5 Extending the pattern language

In this section we briefly discuss possible extensions of the pattern language. Let us first observe that we can add wildcard to our language for free, that is, we can costlessly allow nodes in patterns that do not have a specified label and can match a tree node with any label. Indeed, our automaton can simply guess the label for each processed wildcard, and then proceed as before.

A more interesting extension is to add horizontal relations. Patterns with horizontal relations are defined just like $\{\downarrow, \downarrow^+\}$-patterns we have seen so far, except that they have two additional kinds of edges, denoted by $\rightarrow$ and $\rightarrow^+$, and interpreted respectively as the next sibling and the following sibling.

As soon as we add the next sibling relation, the consistency problem becomes NP-hard. A reduction can be obtained via a simple modification of the one in Theorem 4. Specifically, it suffices to modify the encoding so that the x nodes, c nodes, and a and b nodes are arranged horizontally, rather than vertically. After this modification the pattern in Figure 2 only uses child relation between x and x' nodes. Given that the only descendants of any x node that have label x' are its children, we can replace the child relation with the descendant relation.

▶ **Theorem 15.** *There is an automaton $\mathcal{A}$ s. t.* $\text{Cons}_{\mathcal{A}}$ *is* NP*-complete for* $\{\downarrow^+, \rightarrow\}$*-patterns.*

When only the following sibling is added, we can get a polynomial algorithm.

▶ **Theorem 16.** *For each automaton $\mathcal{A}$,* $\text{Cons}_{\mathcal{A}}$ *is in* PTime *for* $\{\downarrow^+, \rightarrow^+\}$*-patterns.*

In fact, we can again construct a tree automaton recognizing $\{\downarrow^+, \rightarrow^+\}$-patterns consistent with an automaton $\mathcal{A}$. In the following, we explain the main ideas of this construction.

We first explain how to extend the notion of skeleton. Let $\mathcal{A} = (\Sigma, Q, \delta, F)$ be a tree automaton. We assume that horizontal languages in the automaton are given in disjunctive normal form, that is, for each $(a, q) \in \Sigma \times Q$, the language $\delta(a, q)$ is given by a disjunction of disjunction-free regular expressions. We shall refer to these disjunction-free expressions as *clauses* of $\delta(a, q)$. Note that turning a regular expression into this form usually involves an exponential blow-up, but since the automaton is considered to be fixed, this does not change the complexity bound. In the definition below, a letter-state pair (a, q) is *reachable* if there exists a tree T with label a in the root and a run over T that assigns state q to the root. A state q is reachable if there exists a run on any tree that assigns q to the root. Without loss of generality we can assume that all states of $\mathcal{A}$ are reachable.

▶ **Definition 17.** A $\{\downarrow^+, \rightarrow^+\}$-*skeleton* (in this section, just *skeleton*) for an automaton $\mathcal{A} = (\Sigma, Q, \delta, F)$ is a forest labelled with disjunction-free regular expressions over reachable letter-state pairs from $\Sigma \times Q$ such that

- each label is either a single letter-state pair (non-starred node) or a disjunction-free regular expression of the form e^* (starred node);
- starred nodes have no children;
- for each node of label (a, q) the concatenation of the labels of its children forms a disjunction-free regular expression $w_1 u_1 (e_1)^* v_1 w_2 u_2 (e_2)^* v_2 \dots w_n u_n (e_n)^* v_n w_{n+1}$ such that u_i, v_i are generated by e_i and the projection over Q of $w_1(e_1)^* w_2(e_2)^* \dots w_n(e_n)^* w_{n+1}$ is a clause of $\delta(a, q)$;
- similarly for the concatenation of labels of the roots, except that the projection over Q of $w_1(e_1)^* w_2(e_2)^* \dots w_n(e_n)^* w_{n+1}$ is a *suffix* of a clause of $\delta(a', q')$ for some productive q'.

Additionally, non-starred nodes can be flagged as used.

A skeleton is *reduced* if no letter-state pair repeats on a branch, and the words u_i and v_i in the definition above are all empty. A reduced skeleton has its branching bounded by the size of the clauses of the horizontal languages (which are polynomial in the original representation of the languages), and its height bounded by the number of states of the automaton $\mathcal{A}$. Hence, the set of reduced skeletons is finite and each of them is of size at most exponential in the size of $\mathcal{A}$.

Like for $\downarrow^+$-skeletons, we can reduce skeleton s by repeatedly applying the following rules

(H) remove any non-starred node (together with its subtree) whose label occurs in the regular expression e^* labelling its next or previous sibling;
(V) if u and its descendant v are non-starred and carry the same label, then the subtree rooted at u (excluding u) can be replaced by the subtree rooted at v (excluding v).

The automaton recognizing consistent patterns essentially proceeds like before: it assigns reduced skeletons to nodes of the pattern π in a bottom-up fashion, ensuring that they are consistent with each other. More precisely, a node v gets a skeleton that summarizes a way to satisfy the subpattern of π rooted at v. Note that in this subpattern some nodes are connected to v via $\downarrow^+$-edges, and others via $\rightarrow^+$-edges. Thus, the subpattern talks about a certain subforest, which explains why our skeletons are forests. We always assume that v is mapped to the first root of the skeleton.

Suppose that we want to assign a reduced skeleton to a node v. First, we guess a reduced skeleton for a single-node pattern consisting of v alone. This skeleton has at most one used node. Next, we aggregate it with the skeletons assigned to v's children, one by one, using appropriately adjusted injections. Since v's children are now connected to v via $\downarrow^+$ or $\rightarrow^+$, we need two variants of the notion. In both variants, we add to Definition 12 an item guaranteeing preservation of the sibling order: if $v \rightarrow^+ v'$ in s_k, then $i_k(v) \rightarrow^+ i_k(v')$ in s. In the variant for $\downarrow^+$, we require that the first root of the second skeleton is mapped to a descendant of the first root of s, and in the variant for $\rightarrow^+$, it is mapped into a following sibling of the first root of s.

We close this section by commenting that the reasoning above could be extended to cover limited use of child and next-sibling relations: it can be done for patterns, where the maximal length of paths that do not use $\downarrow^+$-edge is bounded.

6 Conclusions

We have shown that under injective semantics, the consistency problem for tree patterns with respect to a fixed automaton is NP-complete by showing the problem to be NP-hard already for child/descendant patterns with at most two descendant edges per branch. This closes an open problem from [2]. Moreover our result is tight with respect to the result of Kopczynski [10], showing tractability for patterns with at most one descendant per branch.

On the positive side, we have provided a polynomial time algorithm in the case of descendant-only tree patterns. The key ingredient is to show that the set of all patterns that are consistent with a given tree automaton $\mathcal{A}$ is a regular tree language. This language only depends on $\mathcal{A}$ and we can effectively construct a tree automaton $\mathcal{A}_\Pi$ recognizing it. Hence, consistency is equivalent to testing whether the pattern belongs to this language, which can be done in polynomial time. Thus, our algorithm is not only polynomial for fixed $\mathcal{A}$, but also fixed-parameter tractable with the size of $\mathcal{A}$ as the parameter.

The involved constant is essentially the size of the automaton $\mathcal{A}_\Pi$, which is double exponential in the size of $\mathcal{A}$. This may seem suboptimal, since the problem is known to be in

NP even when $\mathcal{A}$ is a part of the input. However, while we are guaranteed to find a witness polynomial in the size of the pattern and the automaton, it may be arbitrarily large with respect to the automaton itself. It happens so that these witnesses can be summarized as objects exponential in the size of the automaton (double exponential complexity comes from handling sets of such summaries), but we can see no way to do better than exponential.

We have also examined patterns with additional features: wildcard can be added effortlessly, but horizontal relations pose more problems. We adapted our techniques to show that one can combine descendant and following-sibling without losing tractability, but combining descendant with next-sibling makes the problem NP-complete (for some automata).

Given that without descendant the problem is known to be tractable [2], this charts out completely the tractability frontier for the consistency of injective tree patterns. A question we find interesting and challenging is which of the tractability results can be extended to patterns that are DAGs, rather then trees. For instance, what is the complexity of the consistency problem for descendant-only DAG patterns?

Acknowledgements. We thank the anonymous referees for their comments. The third author was supported by Poland's National Science Centre grant no. UMO-2013/11/D/ST6/03075.

References

1 Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.
2 Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information. *J. ACM*, 58(1):4, 2010.
3 Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2008.
4 Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Semi-structured data with constraints and incomplete information. In *Description Logics*, 1998.
5 Claire David. Complexity of data tree patterns over XML documents. In *MFCS*, pages 278–289, 2008.
6 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
7 Gösta Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science.* Springer, 1991.
8 Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.
9 Yaron Kanza, Werner Nutt, and Yehoshua Sagiv. Querying incomplete information in semistructured data. *J. Comput. Syst. Sci.*, 64(3):655–693, 2002.
10 Eryk Kopczynski. Trees in trees: Is the incomplete information about a tree consistent? In *CSL*, pages 367–380, 2011.
11 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
12 Maarten Marx. XPath with conditional axis relations. In *EDBT*, pages 477–494, 2004.
13 Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.
14 Frank Neven and Thomas Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Log. Meth. Comput. Sci.*, 2(3), 2006.
15 Peter T. Wood. Containment for XPath fragments under DTD constraints. In *ICDT*, pages 297–311, 2003.