Minimization of Tree Pattern Queries

Wojciech Czerwiński*
University of Warsaw

Wim Martens†
Universität Bayreuth

Matthias Niewerth†
Universität Bayreuth

Paweł Parys*
University of Warsaw

ABSTRACT

We investigate minimization of tree pattern queries that use the child relation, descendant relation, node labels, and wildcards. We prove that minimization for such tree patterns is Σ_2^P-complete and thus solve a problem first attacked by Flesca, Furfaro, and Masciari in 2003. We first provide an example that shows that tree patterns cannot be minimized by deleting nodes. This example shows that the M-NR conjecture, which states that minimality of tree patterns is equivalent to their nonredundancy, is false. We then show how the example can be turned into a gadget that allows us to prove Σ_2^P-completeness.

Keywords

XPath, XML, trees, tree patterns, graph databases, optimization, complexity

1. INTRODUCTION

Tree-structured data is among us in many forms: JSON, XML, our filesystems, the secondary structure of RNA, and parse trees for linguistic data, just to name a few examples. Tree pattern queries are a fundamental tool for querying and selecting nodes in tree-structured data. They are present in most query languages for tree-structured data, most notably, XPath [33]. In fundamental research they appear in a wide range of topics. For example, they form a basis for conjunctive queries over trees [22], for models of XML with incomplete information [5], and for the closely related pattern-based XML queries [20]. They are used for specifying guards in Active XML systems [1] and for specifying schema mappings in XML data exchange [4]. Beyond trees, they are a natural language for querying graph databases [14, 27].

Optimization of tree pattern queries is therefore a very natural question. Not only do new results in this direction give us new insights on query optimization in many practical languages, they can also give us a better understanding of the foundations of the above mentioned models which are based on tree patterns. Tree pattern query optimization already attracted significant attention in the form of *query containment* [29, 31, 14], *satisfiability* [6], and *minimization* [3, 13, 18, 25, 32, 36].

In this paper we study the minimization problem for tree pattern queries that use the *child relation*, *descendant relation*, *label tests*, and *wildcards* (henceforth: tree patterns). These tree patterns are widely studied [17, 29, 37, 18, 25, 1, 8, 4, 34] but their minimization problem remained elusive. In particular, the complexity of minimization is unknown.

It is believed that a key in understanding tree pattern minimization lies in understanding the relationship between *minimality* (M) and *nonredundancy* (NR) [18, 25]. Here, a tree pattern is minimal if it has the smallest number of nodes among all equivalent tree patterns. It is nonredundant if none of its leaves or branches can be deleted while remaining equivalent. The question is if minimality and nonredundancy are the same:

> M-NR Conjecture (page 35 of [18], rephrased):
> A tree pattern is minimal if and only if it is nonredundant.

Clearly, every minimal tree pattern is nonredundant, so one direction of the M-NR conjecture trivially holds. The opposite direction is much less clear. If it would be true, it means that, for a given tree pattern p, a minimal tree pattern is always a substructure of p. It would also mean that tree pattern minimization can be solved by tree pattern containment. Indeed, one would be able to minimize tree patterns p by iteratively removing leaves and testing if the obtained tree pattern p' is still contained in p. If no leaf can be removed anymore, the remaining tree pattern would be nonredundant and therefore minimal. Since testing containment of tree patterns is coNP-complete [29], this would mean that one could solve minimization with a polynomial-time algorithm with a coNP oracle and that the minimization problem for tree patterns is coNP-complete.

The minimization problem for tree patterns is not entirely uncontroversial. It was claimed to be coNP-complete in 2003 [17] but the algorithm relied on the M-NR conjecture.

*Supported by Poland's National Science Centre grant no. UMO-2013/11/D/ST6/03075.

†Supported by grant number MA 4938/2–1 from the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe).

PODS'16, June 26-July 01, 2016, San Francisco, CA, USA

ISBN 978-1-4503-4191-2/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2902251.2902295

Kimelfeld and Sagiv [25] proved that, in contrast to claims in [17], the M-NR conjecture is open and, as a consequence, the algorithms from [17] were revised in [18]. The updated work [18] proves that the M-NR conjecture holds for tree patterns in which *every wildcard node has at most one child* and presents a coNP algorithm for this case. It is needless to say that [18, 25] contain a wealth of valuable results on tree patterns and their minimization (many of which we use here), but the most central questions, that is, the status of the M-NR conjecture and the question of the complexity of tree pattern minimization, remained open.

Our main contributions are the following:

- We prove that the M-NR conjecture is false by providing a tree pattern that is nonredundant but not minimal.
- We prove that tree pattern minimization is Σ_2^P-complete. This means that, unless $\Sigma_2^P = \text{coNP}$, we have that tree pattern minimization *cannot* be solved by a polynomial-time algorithm with an oracle for tree pattern containment.
- Interestingly, our counterexample and our gadgets in the Σ_2^P-hardness proof use only two wildcard nodes with two children, which is only barely beyond the fragment for which the M-NR conjecture is known to hold.

The Bigger Picture.

This paper fits naturally in a line of research that originated in the early days of database theory. Query minimization and optimization by removing redundant parts goes back to the seminal work of Chandra and Merlin [11] and has since then been successfully adopted for many types of queries, in various data models such as relations and trees (see, e.g., [3, 10, 11, 18, 32, 36]). In this section we highlight a few parallels and differences between conjunctive queries and acyclic conjunctive queries over relational data and over tree-structured data.

Conjunctive Queries over Relations. Minimization, containment, and evaluation of conjunctive queries over relations are well known to be NP-complete [11]. For *acyclic* conjunctive queries, which have been extensively studied (see, e.g., [12, 23, 38]), the complexity of these problems is in polynomial time. **Conjunctive Queries over Trees.** Conjunctive queries over trees [22] are different from conjunctive queries over relations in two respects. First, the underlying model is based on trees instead of relations and second, conjunctive queries over trees use different built-in relations. Most notably, apart from the child relation, they can use the descendant relation and therefore have the power to reason about certain transitive closures. They therefore query a more restricted data model than their counterpart over relations but in return they have more powerful reasoning. The original definition of conjunctive queries over trees allows for many built-in relations (child, descendant, next-sibling, following-sibling, etc. [22]); we only discuss the two most basic ones, child and descendant, in the following.

Conjunctive queries over trees have an NP-complete evaluation problem [22] just like conjunctive queries over relations. Their containment problem, however, is Π_2^P-complete [7] and the complexity of their minimization problem is unknown.

Tree patterns can be seen as acyclic variants of conjunctive queries over trees. Their evaluation problem is in polynomial time [21] and their containment problem is coNP-complete [29]. By proving that minimization of tree patterns is Σ_2^P-complete, we finish a natural step in this line of research.

Tree Patterns as a Graph Query Language.

Due to their modal nature, tree patterns and XPath-like languages are also suitable languages for querying graph databases [9, 24, 27, 2, 28, 19]. In fact, the complexity of tree pattern containment does not depend on whether they are evaluated over trees or over graphs, see [29, Section 5.3] and [14, Section 7]. The same is true for the minimization problem. Therefore, the complexity results in this paper can be extended to tree patterns over graphs as well. We present all results in terms of trees because it makes proofs considerably simpler.

The problems for trees can even be extended to *data graphs* [27] and patterns that compare data values with constants (such comparisons are essentially the same as the label tests of tree patterns). However, as soon as data value *comparisons* enter the picture, such a straightforward extension to graphs does not work anymore, see e.g. [26].

Outline.

We present the counterexample to the M-NR conjecture in Section 3. We also show in Section 3 that minimal tree patterns are not unique (up to adornments). In Section 4 we prove that tree pattern minimization is Σ_2^P-complete. We discuss implications on k-ary queries and further outlooks in Section 5.

2. PRELIMINARIES

We are interested in finite, labeled, unordered trees.[1] A *labeled unordered tree* is a triple (V, E, lab), where V is a finite nonempty set of *nodes*, E is a set of *edges* $(u, v) \in V \times V$ and $\text{lab} : V \to \Sigma$ is a *labeling function* assigning to every node its label coming from an infinite set of labels Σ. If $(u, v) \in E$ then we say that u is the *parent* of v and v is a *child* of u. We demand that for every node v there is at most one $(u, v) \in E$, so in trees the parent is uniquely determined. There is a unique node without parent, which we denote $\text{root}(t)$ and call the *root* of t. The *descendant* and *ancestor* relations are transitive closures of the child and parent relations, respectively. We say that a child of a node u is a 1-*descendant* of u and a child of a k-*descendant* of u is a $(k+1)$-*descendant* of u for any $k \in \mathbb{N}$. We define k-*ancestors* similarly. A node has *depth* k if it is a k-descendant of the root. In the sequel we just use the term *trees* for referring to labeled unordered trees.

For a tree $t = (V, E, \text{lab})$ and a node $v \in V$ we denote by t^v the subtree of t rooted in node v. By $t \setminus v$ we denote the tree obtained from t by deleting the subtree rooted at v (including node v itself).

A *tree pattern* is intended to describe a set of trees. It is a special type of a tree; its set of edges is divided into two disjoint sets: *child edges* and *descendant edges* (we draw descendant edges using double lines). Its labeling function

[1] However, we don't require trees to be unordered. Our results are the same for ordered trees. Tree patterns, however, are inherently unordered.

provides every node with a label from Σ or a special label $*$ which we assume not to be in Σ and call *wildcard*. We denote $\Sigma_* = \Sigma \cup \{*\}$. The intended meaning of the wildcard is not to specify any particular label. For a tree pattern $p = (V_p, E_p, \mathrm{lab}_p)$ and a tree $t = (V, E, \mathrm{lab})$, a function $\pi : V_p \to V$ is a *strong embedding*[2] of p in t if it fulfills all the following conditions:

(1) if $\mathrm{lab}_p(v) \neq *$ for $v \in V_p$ then $\mathrm{lab}_p(v) = \mathrm{lab}(\pi(v))$,

(2) if $(u, v) \in E_p$ is a child edge then $\pi(u)$ is a parent of $\pi(v)$ in the tree t,

(3) if $(u, v) \in E_p$ is a descendant edge then $\pi(u)$ is an ancestor of $\pi(v)$ in the tree t, and

(4) $\pi(\mathrm{root}(p)) = \mathrm{root}(t)$.

We say that p *strongly embeds* in t if there exists a strong embedding of p in t. We say that π is a *weak embedding* of p in t and p *weakly embeds* in t if the above conditions (1)–(3) are fulfilled, but not necessarily $\pi(\mathrm{root}(p)) = \mathrm{root}(t)$. Figure 1 contains examples of strong and weak embeddings. Notice that we do not require embeddings to be injective (Figure 1(c)).

Equivalence, Containment, and Minimality.

The *(strong) language of a tree pattern* p, denoted by $L_S(p)$, is the set of trees in which p strongly embeds. A tree pattern p_1 is *(strongly) contained* in a tree pattern p_2 if $L_S(p_1) \subseteq L_S(p_2)$, which we denote by $p_1 \subseteq_S p_2$. If $p_1 \subseteq_S p_2$ and $p_1 \supseteq_S p_2$ then we say that the tree patterns p_1 and p_2 are *(strongly) equivalent* and we write $p_1 \equiv_S p_2$.

We call a tree pattern p *redundant* if one of its nodes can be removed without changing its language. More formally, p is redundant if it is strongly equivalent to $p \setminus v$ for a node v of p. In this case, v is a *redundant node*. If p is not redundant we say that it is *nonredundant*. It is known that a pattern is redundant if and only if it has a redundant leaf [25, Proposition 3.3].

The *size* of a tree pattern p, denoted $\mathrm{size}(p)$, is the number of its nodes. A tree pattern p is said to be *minimal* if there is no tree pattern p' that is equivalent to p but has strictly smaller size.

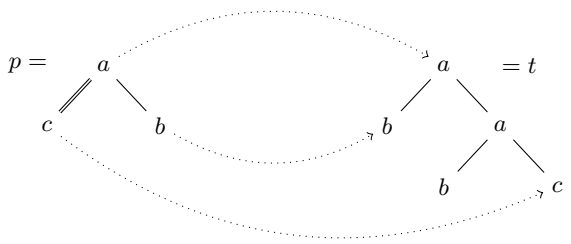
Analogously, we define *weak language*, *weak containment*, *weak equivalence*, *weak redundancy*, and *weak nonredundancy*. The definitions are exactly the same, but use weak embeddings instead of strong embeddings. The notation for weak containment and weak equivalence is $p_1 \subseteq_W p_2$ and $p_1 \equiv_W p_2$, respectively. It is well-known that containment of tree patterns, i.e., deciding for two given tree patterns p and q if $p \subseteq_S q$, is coNP-complete.

THEOREM 2.1 ([29]). *Containment of tree patterns is* coNP*-complete.*

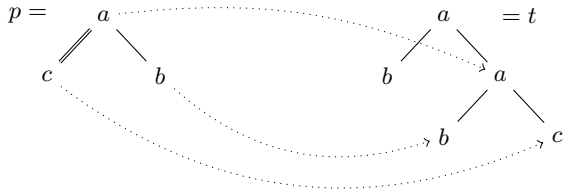
We now mention a weak version of the M-NR Conjecture which was proved in [18]. We require the following definition.

DEFINITION 2.2 ($*$-NARROW PATTERN). *A tree pattern is a* $*$-narrow pattern *if every wildcard node has at most one child.*

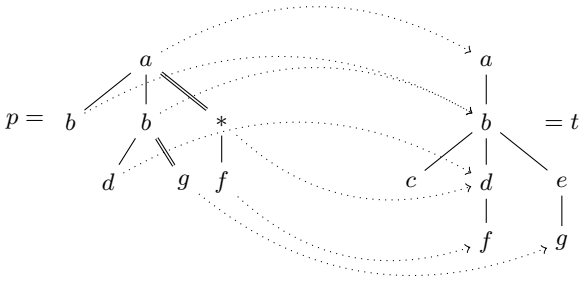
[2]A strong embedding is sometimes also called *root embedding*.



(a) A strong embedding of a tree pattern p in a tree t.



(b) A weak embedding of a tree pattern p in a tree t.



(c) A strong embedding of a tree pattern p in tree t. Embeddings are not required to be injective.

Figure 1: Examples of strong and weak embeddings.

For example, the tree patterns in Figure 1 are $*$-narrow patterns.

LEMMA 2.3 (COROLLARY 4.5 IN [18]). *If p is a $*$-narrow pattern, then p is minimal if and only if p is nonredundant.*

Canonical trees.

Canonical trees were introduced by Miklau and Suciu [29] for studying the containment problem for tree patterns. We need them in our paper to simplify proofs.

Let $z \in \Sigma$ be a special label that does not occur in any tree pattern that we consider in the paper. (We can assume that such a label exists because Σ is infinite.) A *canonical tree* of a tree pattern p is a tree obtained from p by application of the two following steps:

- for every node v such that $\mathrm{lab}_p(v) = *$, we relabel $\mathrm{lab}(v) = z$,
- change every descendant edge in p to a (nonempty) sequence of edges in t in such a way that all newly created nodes are labeled by z.

Notice that it is possible in the last step that no new nodes are created. This happens when each descendant edge is

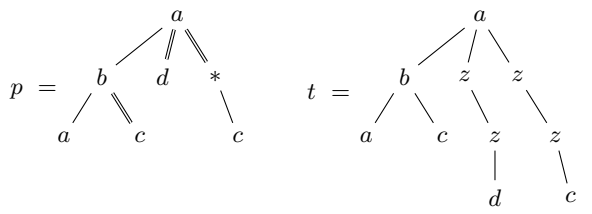


Figure 2: A tree pattern p and a canonical tree t.

replaced by a single child edge. An example of a canonical tree is given in Figure 2. We denote by $\mathrm{Can}(p)$ the set of all canonical trees of p.

Given a pattern p and one of its canonical trees t, there is a canonical injective embedding of the non-wildcard nodes of p into t. We sometimes use this correspondence to talk about nodes in t. That is, for a non-wildcard node u of p, we use this injective embedding to identify *the node in t corresponding to u*.

LEMMA 2.4 (PROPOSITION 3 IN [29]). *Let p_1 and p_2 be two tree patterns. Then $p_1 \subseteq_S p_2$ if and only if $Can(p_1) \subseteq L_S(p_2)$.*

A corresponding lemma for weak containment can be proved analogously as in [29].

Homomorphisms.

Let $p_1 = (V_{p_1}, E_{p_1}, \mathrm{lab}_{p_1})$ and $p_2 = (V_{p_2}, E_{p_2}, \mathrm{lab}_{p_2})$ be tree patterns. A *homomorphism* from p_1 to p_2 is a function $h : V_{p_1} \to V_{p_2}$ that fulfills the following conditions:

(1) $h(\mathrm{root}(p_1)) = \mathrm{root}(p_2)$,

(2) if $\mathrm{lab}_{p_1}(v) \neq *$ for $v \in V_{p_1}$ then $\mathrm{lab}_{p_1}(v) = \mathrm{lab}_{p_2}(h(v))$,

(3) if $(u, v) \in E_{p_1}$ is a child edge then $(h(u), h(v)) \in E_{p_2}$ is a child edge, and

(4) if $(u, v) \in E_{p_1}$ is a descendant edge then $h(u)$ is an ancestor of $h(v)$ in p_2.

The existence of a homomorphism h from p_1 to p_2 is a sufficient condition for $p_2 \subseteq_S p_1$ [29]. Essentially, the reason is that, if π is a strong embedding of p_2 in a tree t, then $\pi \circ h$ is a strong embedding of p_1 in t. We make use of this fact later in the paper.

3. NONREDUNDANCY AND MINIMALITY

In this section we present a counterexample for the M-NR conjecture. We build further on this example to show that minimal tree patterns are not unique. We choose the examples in such a way that they help the reader to understand the gadgets we use in Section 4.

Nonredundancy ≠ Minimality.

We will show that the left tree pattern p in Figure 3 is nonredundant and not minimal. One thing is easy to see: the tree pattern q on the right is smaller. Seeing that the left tree pattern p is nonredundant and equivalent to the tree pattern q on the right requires more work.

First we show that the tree pattern p is nonredundant. To this end, it suffices to show that none of its leaves can be deleted while remaining equivalent [25, Proposition 3.3]. For the purpose of this proof, we order the leaves in p from left to right, that is, the *first c_1-leaf* is the one in depth 7, the *second c_1-leaf* is the leftmost leaf on depth 8, and so on.

- If the first c_1-leaf is removed, then the resulting tree pattern matches the tree t_1 in Figure 4 by the strong embedding π which we partly illustrated in that figure. However, the tree pattern p does not match. The reason why we cannot remove the first c_2-leaf of p is analogous (replace the first c_2-leaf in t_1 by a c_1-leaf).

- If the second c_1-leaf is removed, then the resulting tree pattern matches the tree t_2 in Figure 4 using the strong embedding π which we partly illustrated in that figure. However, the tree pattern p does not match. The reason why we cannot remove the second c_2-leaf of p is analogous (replace the first c_2-leaf in t_2 by a c_1-leaf).

- Finally, if any of the other c_1- or c_2-leaves would be removed, the tree pattern would match the tree t_3 in Figure 4 in which the corresponding (circled) leaf would be removed and this is always a tree that is not matched by p.

Finally, we show that the tree pattern p is (strongly) equivalent to the tree pattern q. To this end, observe that $q \subseteq_S p$ because q is more restrictive: it has the same requirements as p but, in addition it says that the nodes onto which the second c_1- and c_2-leaves are matched have the same parent. It therefore only remains to show that $p \subseteq_S q$. By Lemma 2.4, it suffices to prove that $\mathrm{Can}(p) \subseteq_S L_S(q)$.

Let $t \in \mathrm{Can}(p)$. Let π_p be a strong embedding of p in t. Consider the nodes u_A and u_B in p. We make a case distinction on the number of nodes between $\pi_p(u_A)$ and $\pi_p(u_B)$ in t and show in each case how to construct a strong embedding π_q from q in t.

- If $\pi_p(u_A)$ is the parent of $\pi_p(u_B)$, then we can define $\pi_q(v_2) := \pi_p(u_4)$. For all non-descendants of v_2, we define π_q the same as π_p.

- If $\pi_p(u_A)$ is the 2-ancestor of $\pi_p(u_B)$ (see Figure 5 left), then we define $\pi_q(v_1) := \pi_p(u_2)$, and $\pi_q(v_2) := \pi_p(u_4)$, and $\pi_q(v_B) := \pi_p(u_B)$, and, since the distance between $\pi_q(v_2)$ and $\pi_q(v_A)$ is four, we map v_A to the parent of $\pi_p(u_B)$. In particular, the leftmost branch of t is not used by π_q at all.

- If $\pi_p(u_A)$ is the 3-ancestor of $\pi_p(u_B)$ (see Figure 5 right), then we define $\pi_q(v_1) := \pi_p(u_B)$ and $\pi_q(v_2) := \pi_p(u_3)$. In particular, the two leftmost branches of t are not used by π_q at all.

- If $\pi_p(u_A)$ is the k-ancestor of $\pi_b(u_B)$ for some $k \geq 4$, we proceed in the same way than in the previous case. Here, we move $\pi_q(v_A)$ downward on the path to $\pi_p(u_B)$ so that the distance between $\pi_q(v_2)$ and $\pi_q(v_A)$ is four.

This means that every canonical tree of p can be (strongly) matched by q. By Lemma 2.4 this means that p and q are strongly equivalent.

We therefore obtained that p is nonredundant but q is equivalent and smaller, which leads to the following theorem:

THEOREM 3.1. *The M-NR conjecture is false.*

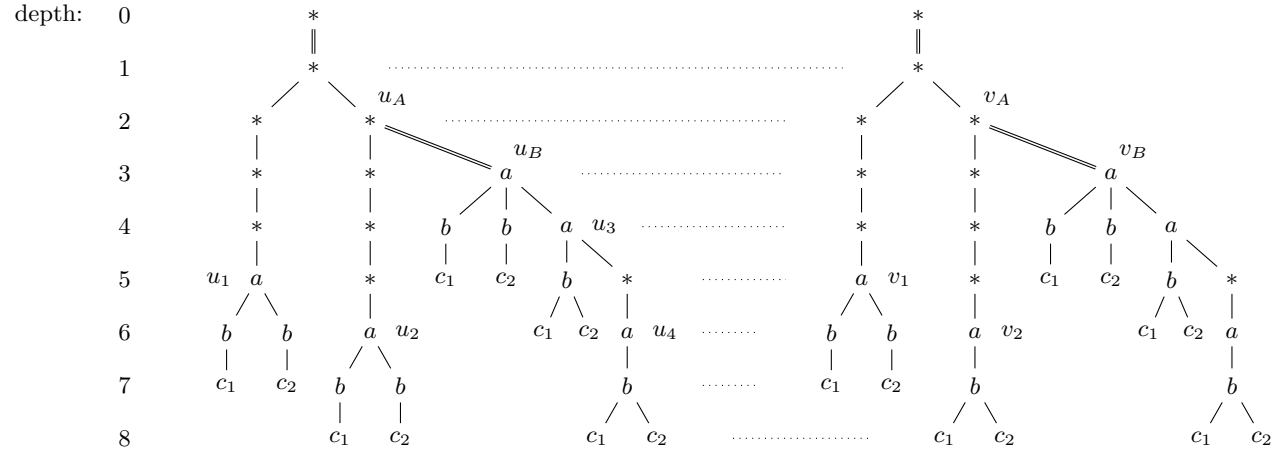


Figure 3: A non-redundant tree pattern p (left) and an equivalent tree pattern q that is smaller (right).

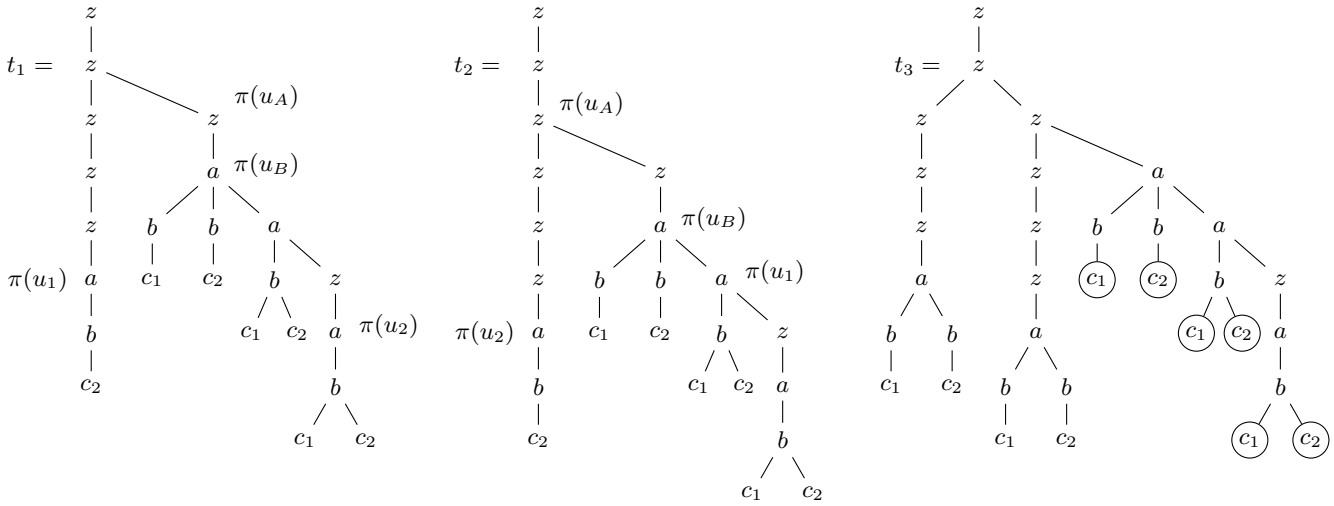


Figure 4: Trees for proving non-redundancy of p in Figure 3.

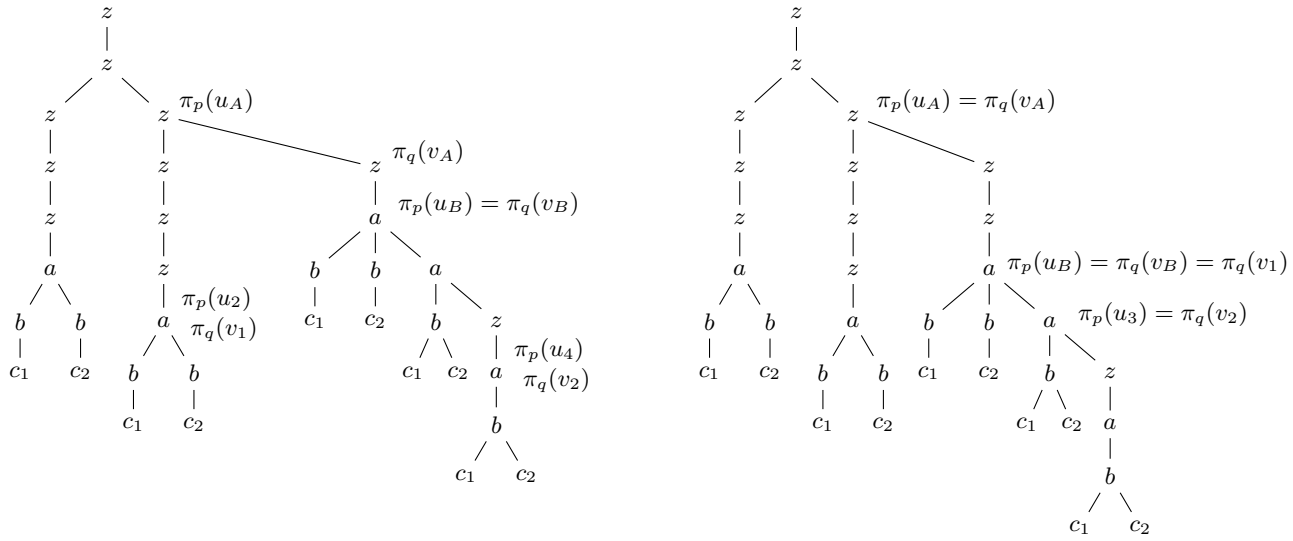


Figure 5: Canonical trees for proving that p and q in Figure 3 are equivalent.

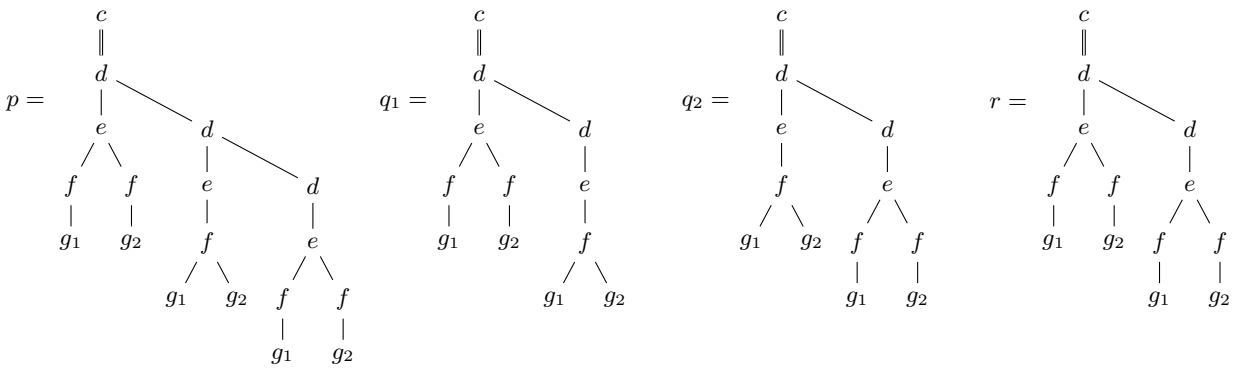


Figure 6: Tree patterns used to show that minimal tree patters can be structurally different.

Minimal Patterns are not Unique up to Adornment.

Minimal tree patterns are not unique. For example, the tree patterns in Figure 7(a) are different, minimal, and both express that there should be at least one node between an a-labeled node and a b-labeled node.

However, it can be argued that the difference between the patterns in Figure 7(a) is rather artificial. In the context of the containment problem for tree patterns, these patterns were used to illustrate that the existence of a homomorphism is not a necessary condition for containment [30, 29]. On the other hand, Milo and Suciu [30] proved that, in restricted cases, it is possible to rewrite patterns in a normal form that alleviates this problem. Miklau and Suciu [29, Section 3.2] extended this normal form to the full class of tree patterns. The patterns in normal form are called *adorned tree patterns*.

Intuitively, adorning a tree pattern corresponds to replacing some paths in the pattern by annotated descendant edges. For example, the adorned pattern equivalent to both patterns in Figure 7(a) is in Figure 7(b). The adornment "≥ 1" means that the pattern requires at least one node between the a-node and the b-node. More formally, Miklau and Suciu define adorned tree patterns as follows. For a given tree pattern, every descendant edge is initially adorned with "≥ 0". Then, adjacent edges sharing a $*$ node are combined into a descendant edge with higher adornment. Only $*$ nodes that have a unique child may be eliminated this way. (If a $*$ node has two or more children, we cannot eliminate it.) The process can also be described as a set of rewrite rules using an XPath-like syntax for tree patterns:

$$\begin{array}{rcl} /\!/ & \to & /\!/^{\geq 0} \\ /\!/^{\geq m} * / & \to & /\!/^{\geq m+1} \\ / * /\!/^{\geq n} & \to & /\!/^{\geq n+1} \\ /\!/^{\geq m} * /\!/^{\geq n} & \to & /\!/^{\geq m+n+1} \end{array}$$

For example, these rules would rewrite $a/\!/*/\!/b/*/c/*/*/*/\!/d$ into $a\,/\!/^{\geq 1}\,b/*/c\,/\!/^{\geq 3}\,d$. In tree pattern syntax, they would rewrite the patterns in Figure 7(a) to the adorned pattern in Figure 7(b).

The question in this section is whether there exist equivalent minimal tree patterns that have different adorned patterns. We show that such patterns can be obtained as follows. Consider the gadget $P(X, Y, Z)$ in Figure 8. For tree patterns p, q, and r, denote by $P(p, q, r)$ the tree pattern obtained from P by instantiating the subtrees marked X, Y, and Z by p, q, and r, respectively. Consider the tree patterns p, q_1, q_2, and r from Figure 6. Then, we claim that

$$P(p, q_1, r) \text{ and } P(p, q_2, r)$$

are equivalent and minimal, but they have different adorned patterns. In fact, notice that adornment does not change anything in $P(p, q_1, r)$ or $P(p, q_2, r)$. Since $P(p, q_1, r)$ is different from $P(p, q_2, r)$, it is immediate that their adornments are also different. It remains to show that $P(p, q_1, r)$ and $P(p, q_2, r)$ are equivalent and minimal, which is non-trivial.

We first show that the tree patterns are equivalent. To this end, we first prove a lemma that already gives insight to a central property of the gadget in Figure 8.

LEMMA 3.2. *Let α, β_1, β_2, and γ be tree patterns that do not use labels in $\{a, b\}$ and such that $\alpha \subseteq_S \beta_1 \subseteq_S \gamma$ and $\alpha \subseteq_S \beta_2 \subseteq_S \gamma$. Then $P(\alpha, \beta_1, \gamma) \equiv_S P(\alpha, \beta_2, \gamma)$.*

PROOF SKETCH. The proof follows the same lines as the proof showing that the two tree patterns in Figure 3 are equivalent. Given a tree t and embedding π_1 of $P(\alpha, \beta_1, \gamma)$ in t, the corresponding embedding π_2 of $P(\alpha, \beta_2, \gamma)$ in t can be constructed using the same case distinction as for tree patterns in Figure 3 and in an analogous manner. Proving that $P(\alpha, \beta_2, \gamma) \subseteq_S P(\alpha, \beta_1, \gamma)$ is analogous. □

We are now ready to show the equivalence between the tree patterns.

PROPOSITION 3.3. *$P(p, q_1, r)$ and $P(p, q_2, r)$ are strongly equivalent.*

PROOF SKETCH. The inclusions $q_1 \subseteq_S r$ and $q_2 \subseteq_S r$ are trivial: q_1 and q_2 restrict r by additionally requiring that two nodes labeled g_1 and g_2 should have the same parent. The inclusion $p \subseteq_S q_1$ holds since q_1 is a strict subpattern of p and the inclusion $p \subseteq_S q_2$ holds since there is a homomorphism π from q_2 to p that maps the topmost d-node of q_2 to the middle d-node of p. The equivalence then follows from Lemma 3.2. □

Minimality of $P(p, q_1, r)$ and $P(p, q_2, r)$ is more technical to prove and requires material which we develop in Section 4. More precisely, minimality of $P(p, q_1, r)$ and $P(p, q_2, r)$ follows from Lemma 4.2. We briefly explain why. Since p and r are $*$-narrow, their minimality is immediate because they are nonredundant (Lemma 2.3). It remains to prove that q_1 and q_2 are smallest tree patterns such that $p \subseteq_S q_i \subseteq_S r$

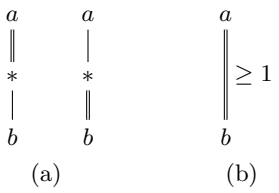


Figure 7: Two minimal but different tree patterns and their equivalent adorned pattern.

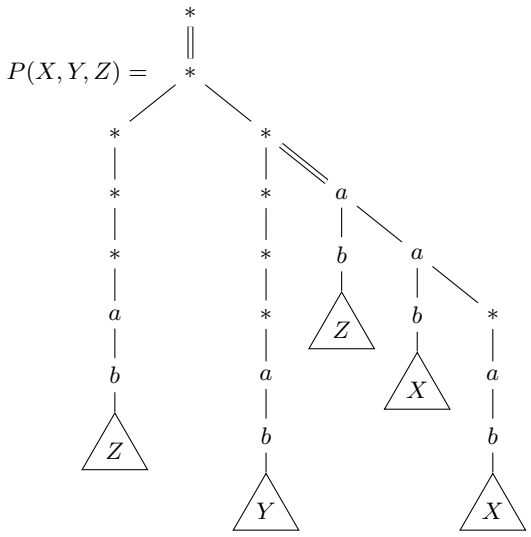


Figure 8: Gadget for constructing tree patterns (Lemma 4.3).

holds. The intuitive[3] reason is that, starting with tree pattern r, we can only get a smaller pattern q that still satisfies $q \subseteq_S r$ by merging f-nodes that are siblings. Furthermore, we can only merge one pair of f-nodes, because otherwise the pattern does not satisfy $p \subseteq_S q$ any more.

4. THE COMPLEXITY OF MINIMIZATION

In this section we study the complexity of the following problems.

Tree Pattern Minimization	
Given:	A tree pattern p and $k \in \mathbb{N}$
Question:	Does there exist a tree pattern q such that size$(q) \leq k$ and $q \equiv_S p$?

Minimality	
Given:	A tree pattern p
Question:	Is p minimal?

It was proved in [18, Theorem 5.9] that Tree Pattern Minimization is coNP-hard. Minimality is known to be NP-hard [25, Theorem 6.3]. The central theorem of this section is:

[3]A more formal proof would be analogous to the proof of Claim 4.7.

Theorem 4.1. Tree Pattern Minimization *is* Σ_2^P*-complete.*

Since testing non-redundancy of a tree pattern is NP-complete [25, Theorem 6.3], the above theorem proves that, under standard complexity theoretic assumptions, the problem Tree Pattern Minimization is more difficult than testing non-redundancy.

We will now prove Theorem 4.1. The Σ_2^P upper bound is straightforward: given an instance of Tree Pattern Minimization consisting of tree pattern p and $k \in \mathbb{N}$, one can guess a tree pattern q of size at most k and test if it is equivalent to p. Since we make polynomially many nondeterministic guesses and since testing equivalence is coNP-complete, this is a Σ_2^P algorithm.

To show hardness, we will introduce an intermediate problem called *relative minimization*. We will give two reductions: the first is from relative minimization to tree pattern minimization and the second one shows that relative minimization is Σ_2^P-hard.

Relative Tree Pattern Minimization	
Given:	Minimal tree patterns p and r such that $p \subseteq_S r$ and $k \in \mathbb{N}$
Question:	Is there a pattern q such that size$(q) \leq k$ and $p \subseteq_S q \subseteq_S r$?

We will use the gadget $P(X, Y, Z)$ from Figure 8. Recall that, for tree patterns p, q, and r, we denote by $P(p, q, r)$ the tree pattern obtained from P by instantiating the subtrees marked X, Y, and Z by p, q, and r, respectively.

The following lemma is the technically most difficult result in the paper. It is crucial for connecting Tree Pattern Minimization with Relative Tree Pattern Minimization and essentially proves two things. First, if one inserts minimal patterns p, q, r in the positions X, Y and Z, then one may still be able to obtain a smaller pattern by changing q. Second, all patterns that are equivalent to $P(p, q, r)$, minimal and satisfy some side-conditions are, in a sense, similar to $P(p, q, r)$. Thus in order to show that $P(p, q, r)$ is minimal can focus only on such similar candidates for being smaller and equivalent one. It is this second part that has a very technical proof.

Lemma 4.2. *Let p, q, and r be tree patterns that have at least one node, do not use labels in $\{a, b\}$, and such that $p \subseteq_S q \subseteq_S r$. Then $P(p, q, r)$ is minimal if and only if*

(1) p and r are minimal; and

(2) there is no tree pattern q' such that

- *$p \subseteq_S q' \subseteq_S r$ and*
- *size(q') < size(q).*

Notice that condition (2) in Lemma 4.2 is subtle. If $P(p, q, r)$ is minimal, then q must also be minimal. However, minimality of q does not necessarily imply that $P(p, q, r)$ is minimal. Indeed, if there would be a pattern q' that is not equivalent to q but such that size(q') < size(q) and $p \subseteq_S q' \subseteq_S r$, then $P(p, q', r)$ would be equivalent to $P(p, q, r)$ and smaller.

The proof of the lemma considers an arbitrary pattern that is minimal and equivalent to $P(p, q, r)$ and proves step by step that it must be similar to $P(p, q, r)$. In a second step, we infer that the pattern at Y must be a smallest pattern q

such that $p \subseteq_S q \subseteq_S r$. A particular challenge in the proof is the lack of methods that work for the general class of tree patterns.

LEMMA 4.3. RELATIVE TREE PATTERN MINIMIZATION *is reducible to* TREE PATTERN MINIMIZATION *in logarithmic space.*

PROOF. Consider an arbitrary instance of RELATIVE TREE PATTERN MINIMIZATION consisting of minimal tree patterns p and r such that $p \subseteq_S r$, and a number $k \in \mathbb{N}$. We can assume w.l.o.g. that p and r have at least one node. Since we can rename labels, we can also assume that p and r do not use the labels a or b. We will construct an instance p_m and $k' \in \mathbb{N}$ of TREE PATTERN MINIMIZATION so that p_m has an equivalent pattern of size at most k' if and only if there is a tree pattern q with $\text{size}(q) \leq k$ and $p \subseteq_S q \subseteq_S r$.

Tree pattern p_m is $P(p, p, r)$, where the gadget P is illustrated in Figure 8. We define k' as $k + 2|p| + 2|r| + 20$.

We now prove that the reduction is correct. We need to prove two implications. For the first, assume that p, r, and k have a solution q w.r.t. RELATIVE TREE PATTERN MINIMIZATION. In this case we know from Lemma 3.2 that $p_m = P(p, p, r) \equiv_S P(p, q, r)$. Furthermore, the size of $P(p, q, r)$ is $\text{size}(q) + 2|p| + 2|r| + 20 \leq k'$.

We prove the other implication. We assume that $p_m = P(p, p, r)$ has an equivalent pattern of size at most k' and we want to prove that there exists a pattern q of size at most k such that $p \subseteq_S q \subseteq_S r$. Let q be a smallest pattern such that $p \subseteq_S q \subseteq_S r$ (more precisely, there exists no pattern q' such that $\text{size}(q') < \text{size}(q)$ and $p \subseteq_S q' \subseteq_S r$).

By Lemma 3.2, we have that $P(p, q, r) \equiv_S p_m$. Tree patterns p and r are minimal and q is a smallest pattern in the set $\{q' \mid p \subseteq_S q' \subseteq_S r\}$. Therefore, by Lemma 4.2, pattern $P(p, q, r)$ is minimal. Therefore its size is at most $k' = k + 2|p| + 2|r| + 20$. This implies that $|q| \leq k$ and concludes the proof. □

To prove Theorem 4.1 it therefore only remains to prove that RELATIVE TREE PATTERN MINIMIZATION is Σ_2^P-complete, which we do next. We will reduce from the following problem, which is a mild variation of the canonical satisfiability problem of quantified $\exists\forall$-formulas ($\exists\forall$-QBF).

∃-VALIDITY	
Given:	A set of pairs of conjunctive clauses $\{(c_1^1, c_1^2), \ldots, (c_m^1, c_m^2)\}$ over variables $x_1, \ldots, x_n$
Question:	Is there a $(i_1, \ldots, i_m) \in \{1, 2\}^m$ such that $c_1^{i_1} \vee \cdots \vee c_m^{i_m}$ is true for every valuation of $x_1, \ldots, x_n$?

Due to the similarity between ∃-VALIDITY and $\exists\forall$-QBF, it is not surprising that ∃-VALIDITY is Σ_2^P-complete.

LEMMA 4.4. ∃-VALIDITY *is* Σ_2^P*-complete.*

PROOF. Membership in Σ_2^P is obvious. For the other direction let

$$\Psi = \exists x_1, \ldots, x_n \forall y_1, \ldots, y_\ell \; \Phi(x_1, \ldots, x_n, y_1, \ldots, y_\ell)$$

be a QBF formula such that $\Phi = c_1 \vee \cdots \vee c_m$ is quantifier-free and in disjunctive normal form. We compute the ∃-VALIDITY instance

$$\{(c_i, c_i) \mid i \in [1, m]\} \cup \{(x_i, \neg x_i) \mid i \in [1, n]\} .$$

For the correctness of the reduction, we first observe, that the formula Ψ is equivalent to

$$\Psi' = \exists x_1, \ldots, x_n \forall z_1, \ldots, z_n, y_1, \ldots, y_\ell \\ \Phi(z_1, \ldots, z_n, y_1, \ldots, y_\ell) \vee z_1 \neq x_1 \vee \cdots \vee z_n \neq x_n .$$

Now it is easy to see the correctness, as the pairs

$$(c_1, c_1), \ldots, (c_m, c_m)$$

enforce that each original clause has to be satisfied (there is no choice) and the pairs $(x_1, \neg x_1), \ldots, (x_n, \neg x_n)$ allow an existential choice for the values of the x-variables as demonstrated in Ψ', i.e., if some x-variable x_i should be true, we choose $\neg x_i$ from the pair $(x_i, \neg x_i)$ and vice versa. □

We now use ∃-VALIDITY to prove that RELATIVE TREE PATTERN MINIMIZATION is Σ_2^P-complete, which is our final step in proving Theorem 4.1.

LEMMA 4.5. RELATIVE TREE PATTERN MINIMIZATION *is* Σ_2^P*-complete.*

PROOF. The upper bound follows from the straightforward algorithm: guess q and check whether $p \subseteq_S q \subseteq_S r$. Clearly, guessing q can be done by a polynomial number of guesses and the containment tests can be done in coNP by Theorem 2.1.

For the lower bound, we reduce from ∃-VALIDITY. We build on Miklau and Suciu's proof that containment of tree patterns is coNP-hard ([29, Proofs of Lemma 3 and Theorem 4]) and extend their idea. Let $I = \{(c_1^1, c_1^2), \ldots, (c_m^1, c_m^2)\}$ be an instance of ∃-VALIDITY. We compute the patterns p and r as given in Figure 9. We let $k = |r| - m$. Notice that the pattern p only depends on the *number* of clauses and variables of I (it uses m in the picture of p and n in the subpatterns C and D) and not on the clauses themselves. Furthermore, p does not contain any wildcards and only contains descendant edges in its subquery D. Pattern r does contain wildcards in the subpatterns C_i^j, but these wildcards have only one child. Therefore, p and r are $*$-narrow patterns.

The idea of p and r is that each subpattern of r that is rooted at a b-labeled node represents a pair of clauses and the subpattern C_i^j represents the clause c_i^j for each $i \in \{1, \ldots, m\}$ and $j \in \{1, 2\}$. The subpattern C_i^j has a root labeled g. For each positive literal x_i of c_i^j, the g-labeled node has an x_i-labeled child that itself has an x_i-labeled child, connected by a child edge. For each negative literal $\neg x_i$ of c_i^j, the g-labeled node has an x_i-labeled child, that has a wildcard node as child which has an x_i-labeled child connected by a descendant edge.

Since the only descendant edges of p occur in the subpattern D, the canonical trees of p only differ from p in the subtree corresponding to D.

We will show that I is a true instance of ∃-VALIDITY if and only if p, r, and k are a true instance of RELATIVE TREE PATTERN MINIMIZATION. To this end, we first present two claims and prove correctness of the reduction based on these claims.

The first claim states that p, r, and k are an instance of RELATIVE TREE PATTERN MINIMIZATION.

CLAIM 4.6. *The tree patterns p and r are minimal.*

We prove the claim. Since p and r both are $*$-narrow, it suffices to show that both patterns are nonredundant, according to Lemma 2.3. It is easy (but tedious) to verify that

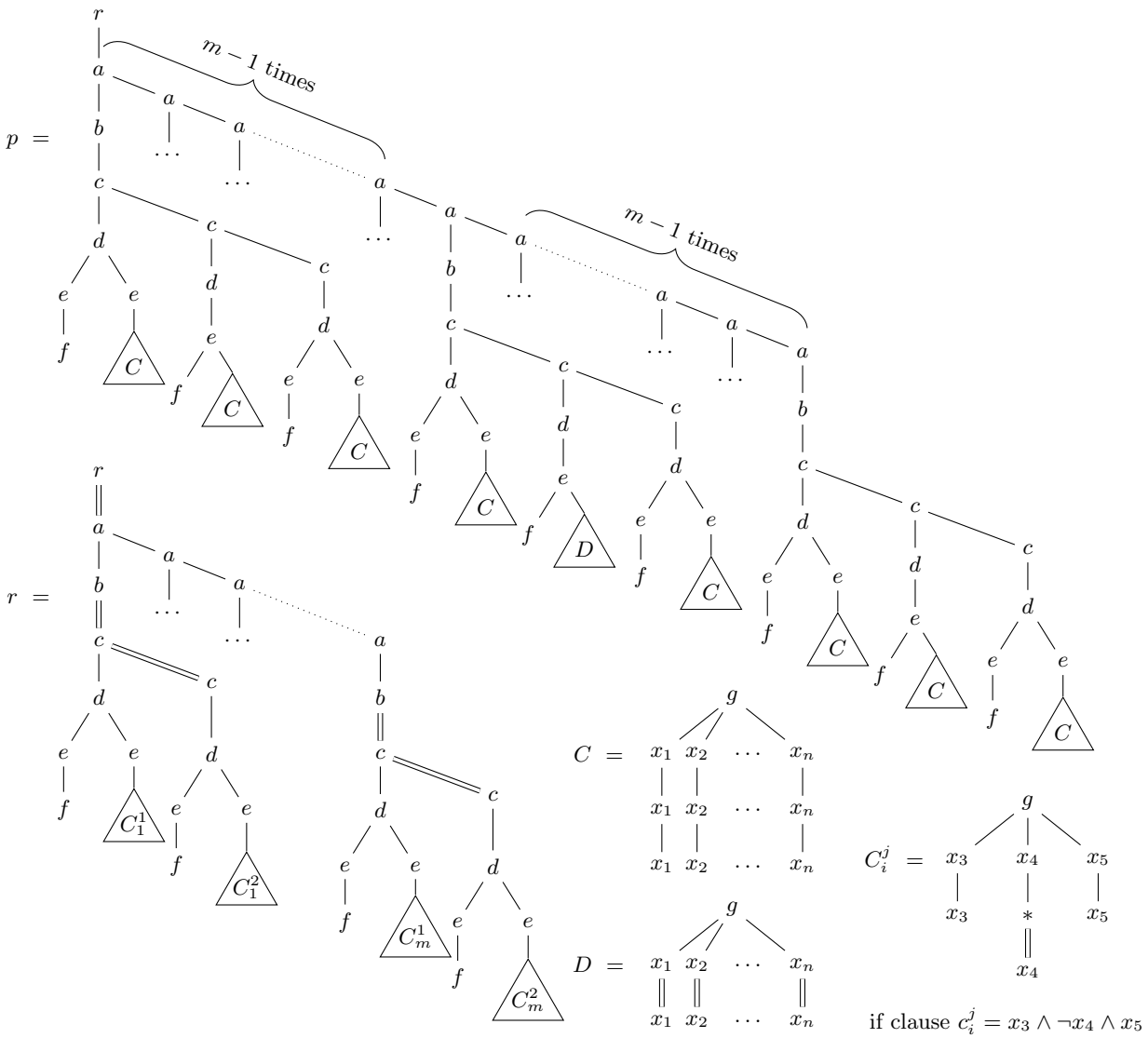


Figure 9: Patterns used in the proof of Theorem 4.1

removing any leaf from any canonical tree of p results in a tree that is not in $L(p)$. By Lemma 2.4, this means that none of the patterns obtained from p by removing a leaf is equivalent to p. This means that p is nonredundant and therefore also minimal. The proof for r is analogous. This concludes the proof of Claim 4.6.

The second claim limits the form of solutions q to the instance. We say that a tree pattern q is in *normal form* if it can be obtained from pattern r by the following algorithm: Select in each subpattern rooted at a b-labeled node at most one subpattern rooted at a d-labeled node. In each selected subpattern, merge the two nodes labeled e — the other nodes remain unchanged. Furthermore, we allow to replace descendant edges by child edges.[4]

[4]The important operation, however, is the merging of nodes, since this operation changes the size of a pattern. The only reason why we also allow replacement of descendant edges by child edges is because we cannot avoid it in Claim 4.7.

Claim 4.7. *Let q be a smallest pattern in the set $\{q' \mid p \subseteq_S q' \subseteq_S r\}$. Then q is in normal form.*

We sketch a proof of the claim. Consider a fixed pattern q among the smallest (possibly non-equivalent) patterns in the set

$$\{q' \mid p \subseteq_S q' \subseteq_S r\}\,.$$

Let t_q be the (unique) smallest canonical tree of q and π be an embedding of r into t_q. We note that t_q contains no information about which edges of q are descendant edges.

We first show that π is surjective. Indeed, if this is not the case, then we can construct a tree pattern q' such that $p \subseteq_S q' \subseteq_S r$ and $\text{size}(q') < \text{size}(q)$ as follows. We remove a node of t_q not used by π, relabel every z-node as wildcard node, and change every edge $(\pi(u), \pi(v))$ of t_q for which (u, v) is a descendant edge in r into a descendant edge. We have that $p \subseteq_S q \subseteq_S q'$, as the construction only removes restrictions. Furthermore, we also have that $q' \subseteq_S r$ since

there are no nodes u, v and w in r such that all of the following hold:

- v is a descendant of u;
- (u, w) is a descendant edge;
- v and w have the same label or one of the nodes is a wildcard; and
- v and w are in different subtrees, i.e., v is not a descendant of w or vice versa.

The existence of q' therefore contradicts the definition of q, which means that π is surjective.

The claim now follows from the three observations below:

(1) For each child edge (u, v) of r, the edge $(\pi(u), \pi(v))$ corresponds to a child edge of q, because otherwise we would have $p \not\subseteq_S q$. We note that it might be possible that, for some descendant edge (u, v) of r, the edge $(\pi(u), \pi(v))$ corresponds to a child edge of q.

(2) If for two nodes v and w of r we have that $\pi(v) = \pi(w)$, then both nodes have label e and are siblings. All other possibilities can be excluded by considering the relative depth of nodes in the pattern and ancestor-descendant relationships.

(3) Furthermore, in any b-subpattern, there is at most one pair of e-labeled siblings such that $\pi(v) = \pi(w)$, since otherwise we would have that $p \not\subseteq_S q$.

Indeed, from (1) and (2) we can conclude that q can be obtained from r by merging e-labeled siblings and possibly replacing some descendant edges by child edges. From (3) we can conclude that in each b-subpattern it suffices to merge one pair of e-labeled siblings, which concludes the proof of Claim 4.7.

We now proceed with the proof of Lemma 4.5. We say that a pattern q is a *solution* to I if it is has size k, satisfies $p \subseteq_S q \subseteq_S r$, and is in normal form, i.e., in every subpattern rooted at a b-labeled node, two e-labeled siblings are merged.

Let q be a tree pattern of size k in normal form. We denote by v_i^j for $i \in \{1, \ldots, m\}$ and $j \in \{1, 2\}$ the d-labeled node of q that is ancestor of the C_i^j subpattern. We define a function $\mathrm{f}^q : \{1, \ldots, m\} \to \{1, 2\}$ as follows:

$$\mathrm{f}^q(i) = \begin{cases} 1 & \text{if } v_i^1 \text{ has exactly one child} \\ 2 & \text{if } v_i^2 \text{ has exactly one child} \end{cases}$$

Notice that f^q is well-defined if q has size k and is in normal form.

Let t be a canonical tree of pattern p and let $d(x_i)$ denote the distance between the two x_i-labeled nodes in the subtree of t corresponding to the D-subpattern of p. With t we associate a valuation σ^t of the variables $x_1, \ldots, x_n$ as follows:

$$\sigma^t(x_i) = \begin{cases} \text{true} & \text{if } d(x_i) = 1 \\ \text{false} & \text{if } d(x_i) > 1 \end{cases}$$

We can show the following points, which prove the equivalence between tree pattern minimization and ∃-VALIDITY.

(a) If q is a solution to I, then σ^t satisfies

$$c_1^{\mathrm{f}^q(1)} \vee \cdots \vee c_m^{\mathrm{f}^q(m)}$$

for every canonical tree t of r. This shows universality of the formula because there exists a canonical tree t of p such that $\sigma^t = \rho$ for each possible valuation ρ of variables.

For the other direction, let $\iota : \{0, \ldots, m\} \to \{1, 2\}$ be a choice of clauses. We can show the following.

(b) If the formula $c_1^{\iota(1)} \vee \cdots \vee c_m^{\iota(m)}$ is universally true, then the normal form pattern q of size k such that

$$\mathrm{f}^q(j) = \iota(j) \text{ for all } j \in \{1, \ldots, m\}$$

satisfies $r \subseteq_S q \subseteq_S p$.

Statements (a) and (b) show that the reduction to tree pattern minimization is correct. Notice that the patterns p and q and the number k can be computed using logarithmic space.

We omit the proofs of statements (a) and (b). This concludes the proof of Lemma 4.5. □

The techniques we used for proving that TREE PATTERN MINIMIZATION is Σ_2^P-complete can also be used to prove that MINIMALITY is Π_2^P-complete. The proof for minimality uses the same ideas as the proof for minimization. However it is not immediate, as MINIMALITY can be seen as a variant of TREE PATTERN MINIMIZATION for fixed $k = \mathrm{size}(p) - 1$, and thus these two problems are a bit different. The basic observation idea is that we can compute a tree pattern q such that $p \subseteq_S q \subseteq_S r$ and

$$\mathrm{size}(q) = k + 1 = |r| - m + 1 \,.$$

Therefore, $P(p, q, r)$ is minimal if and only if the underlying ∃-validity instance is a false instance.

THEOREM 4.8. MINIMALITY *is* Π_2^P*-complete.*

PROOF SKETCH. Membership in Π_2^P is immediate: the algorithm has to check whether each smaller pattern is non-equivalent. The non-equivalence test can be done in NP since equivalence of tree patterns is coNP-complete [29].

For Π_2^P-hardness, we use essentially the same (combined) reduction as in the proofs of Lemma 4.5 and 4.3. Let I be an instance of ∃-VALIDITY and let p and r be the patterns computed in the reduction from ∃-VALIDITY to RELATIVE TREE PATTERN MINIMIZATION in the proof of Lemma 4.5.

We compute a pattern q from p by merging the e-nodes above the subpatterns C_1^1 to C_{m-1}^1 with their siblings.

It is easy to see that $p \subseteq_S q \subseteq_S r$. The second inclusion holds again, because we restrict the trees by merging nodes. The first inclusion holds, because there exists a homomorphism from q to p which embeds the lowest b-subpattern of q on the b-subpattern containing D. The two c-labeled nodes from the q-pattern can be embedded on the upper and lower c-labeled nodes inside the b-subpattern of p.

Finally, we ask whether the pattern $P(p, q, r)$ is minimal. If the answer is yes, then I has no solution because we already know that a solution to I implies the existence of a pattern q' with $p \subseteq_S q' \subseteq_S r$ and

$$\mathrm{size}(q') = \mathrm{size}(r) - m < \mathrm{size}(q) = \mathrm{size}(r) - m + 1 \,.$$

By Lemma 3.2, we know that $P(p, q', r) \equiv_S P(p, q, r)$. Furthermore, $P(p, q', r)$ is smaller than $P(p, q, r)$.

On the other hand, if there exists a pattern $P_{\min}$ with $P_{\min} \equiv_S P(p, q, r)$ and $\mathrm{size}(P_{\min}) < \mathrm{size}(P(p, q, r))$, then,

by Lemma 4.2, we know that there exists a pattern q' with $\text{size}(q') < \text{size}(q)$ and $p \subseteq_S q' \subseteq r$. As

$$\text{size}(q') < \text{size}(q) = \text{size}(r) - m + 1 \ ,$$

and therefore $\text{size}(q) \leq \text{size}(r) - m$, we know from the reduction in Lemma 4.5, that I has a solution. □

5. DISCUSSION AND OUTLOOK

Boolean versus k-ary Queries.

We proved that minimization for Boolean tree patterns is Σ_2^P-complete. This result can also be extended to k-ary tree patterns (as considered in [29, 25]). However, the technique to transfer this result is not the usual one from [25, Section 5] because, as the authors say, it is not clear if the reduction presented there preserves minimality. We can transfer the complexity directly, however. For k-ary queries, the Σ_2^P upper bound follows by applying the naive algorithm and the lower bound follows from attaching all k output nodes to the root of our gadgets.

Lessons for Minimization.

This work gives new insights on how minimal tree patterns may need to be obtained and is the first to give a tight complexity bound for doing so. Even though our main result is a hardness result, we believe that the new insights can be used to develop better tree pattern optimization algorithms. For example, we know that minimization of $*$-narrow tree patterns can always be done by (iteratively) removing leaves [18]. It was long believed that all tree patterns can be minimized in such a way (see, e.g., [17, 18]) but from this paper we now know that this is not the case. In particular, we now know that it may also be necessary to merge nodes.

This is a fact that we can use for developing better heuristics for greedy tree pattern minimization. That is, for a given tree pattern, we can approximate a minimal equivalent pattern by iteratively removing a leaf or merging two nodes and testing if the resulting tree pattern is still equivalent; until no such operation can be done anymore. Notice that this is a polynomial-time algorithm with a subroutine for equivalence tests (which can in general be coNP-complete).

It is important to note that the presented approach is still a heuristic and does not always produce a minimal pattern. For instance, if $\Pi_2^P \neq$ coNP, then we know that minimal patterns cannot always be obtained from a given pattern by removing leaves and merging nodes. Indeed, assume that this would be true. Then consider a non-minimal pattern p_1 and an equivalent pattern p_2 obtained from p_1 by deleting a node or merging two nodes. By the above assumption, we should also be able to go from p_2 to a minimal pattern of p_2 (and thus also of p_1) by deleting and merging nodes. However, this would mean that the above greedy algorithm can be used to decide Tree Pattern Minimization in coNP.

We also believe that the above argument can be strengthened so that the $\Pi_2^P \neq$ coNP condition is not needed. This proof would be based on the concrete patterns we construct in the proof of Lemma 4.5. Essentially, the argument boils down to showing that there are patterns q'' with $p \subseteq_S q'' \subseteq_S r$ from which no smallest pattern in $\{q' \mid p \subseteq_S q' \subseteq_S r\}$ can be reached by merging nodes. This means that, for the pattern $P(p, q'', r)$, it would be necessary to *split* a node in order to reach a pattern of the form $P(p, q', r)$ where q' is one of the smallest patterns in $\{q' \mid p \subseteq_S q' \subseteq_S r\}$. So, a rewriting sequence from a given pattern to an equivalent minimal one may have to perform steps that make patterns larger.

It is an interesting question which (non-trivial) set of operations would be sufficient to guarantee that a minimal pattern can always be obtained by applying a sequence of such operations to the input pattern. This line of thinking leads to questions about query rewriting and axiomatizations for tree pattern equivalence. Ten Cate and Marx [35] studied such axiomatizations for XPath 2.0 (which contains tree patterns as a sublanguage) and present a sound and complete set of axioms for query equivalence, that is, a set of axioms for rewriting patterns into equivalent ones. Fazzinga, Flesca, and Furfaro [15, 16] considered this question for tree patterns and provide a set of axioms complete up to homomorphism containment.

Acknowledgments

We are very grateful to Benny Kimelfeld for insightful discussions and for bringing the tree pattern query minimization problem to our attention.

6. REFERENCES

[1] S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of active XML systems. *ACM Trans. Database Syst.*, 34(4), 2009.

[2] N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *J. Log. Comput.*, 13(6):939–956, 2003.

[3] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *VLDB J.*, 11(4):315–331, 2002.

[4] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

[5] P. Barceló, L. Libkin, A. Poggi, and C. Sirangelo. XML with incomplete information. *J. ACM*, 58(1):4, 2010.

[6] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2008.

[7] H. Björklund, W. Martens, and T. Schwentick. Conjunctive query containment over trees. *J. Comput. Syst. Sci.*, 77(3):450–472, 2011.

[8] H. Björklund, W. Martens, and T. Schwentick. Validity of tree pattern queries with respect to schema information. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 171–182, 2013.

[9] S. Cassidy. Generalizing XPath for directed graphs. In *Extreme Markup Languages Conference*, 2003.

[10] E. P. F. Chan. Containment and minimization of positive conjunctive queries in OODB's. In *Symposium on Principles of Database Systems (PODS)*, pages 202–211, 1992.

[11] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Symposium on Theory of Computing (STOC)*, pages 77–90, 1977.

[12] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

[13] D. Chen and C. Y. Chan. Minimization of tree pattern queries with constraints. In *International Conference on Management of Data (SIGMOD)*, pages 609–622, 2008.

[14] W. Czerwiński, W. Martens, P. Parys, and M. Przybyłko. The (almost) complete guide to tree pattern containment. In *Symposium on Principles of Database Systems (PODS)*, pages 117–130, 2015.

[15] B. Fazzinga, S. Flesca, and F. Furfaro. On the expressiveness of generalization rules for xpath query relaxation. In *International Database Engineering and Applications Symposium (IDEAS)*, pages 157–168, 2010.

[16] B. Fazzinga, S. Flesca, and F. Furfaro. XPath query relaxation through rewriting rules. *IEEE Trans. Knowl. Data Eng.*, 23(10):1583–1600, 2011.

[17] S. Flesca, F. Furfaro, and E. Masciari. On the minimization of XPath queries. In *International Conference on Very Large Data Bases (VLDB)*, pages 153–164, 2003.

[18] S. Flesca, F. Furfaro, and E. Masciari. On the minimization of XPath queries. *J. ACM*, 55(1), 2008.

[19] G. H. L. Fletcher, M. Gyssens, D. Leinders, J. V. den Bussche, D. V. Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. In *International Conference on Database Theory (ICDT)*, pages 197–207, 2011.

[20] A. Gheerbrant, L. Libkin, and C. Sirangelo. Reasoning about pattern-based XML queries. In *International Conference on Web Reasoning and Rule Systems (RR)*, pages 4–18, 2013.

[21] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.

[22] G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006.

[23] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

[24] Gremlin Language. https://github.com/tinkerpop/gremlin/wiki, 2013.

[25] B. Kimelfeld and Y. Sagiv. Revisiting redundancy and minimization in an XPath fragment. In *International Conference on Extending Database Technology (EDBT)*, pages 61–72, 2008.

[26] E. V. Kostylev, J. L. Reutter, and D. Vrgoc. Containment of data graph queries. In *International Conference on Database Theory (ICDT)*, pages 131–142, 2014.

[27] L. Libkin, W. Martens, and D. Vrgoc. Querying graph databases with XPath. In *International Conference on Database Theory (ICDT)*, pages 129–140, 2013.

[28] M. Marx. XPath and modal logics of finite DAG's. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 150–164, 2003.

[29] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.

[30] T. Milo and D. Suciu. Index structures for path expressions. In *International Conference on Database Theory (ICDT)*, pages 277–295, 1999.

[31] F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, 2(3), 2006.

[32] P. Ramanan. Efficient algorithms for minimizing tree pattern queries. In *International Conference on Management of Data (SIGMOD)*, pages 299–309, 2002.

[33] J. Robie, D. Chamberlin, M. Dyck, and J. Snelson. XML Path Language 3.0. Technical report, World Wide Web Consortium, April 2014. Recommendation, http://www.w3.org/TR/2014/REC-xpath-30-20140408/.

[34] S. Staworko and P. Wieczorek. Characterizing XML twig queries with examples. In *International Conference on Database Theory (ICDT)*, pages 144–160, 2015.

[35] B. ten Cate and M. Marx. Axiomatizing the logical core of XPath 2.0. *Theory Comput. Syst.*, 44(4):561–589, 2009.

[36] P. T. Wood. Minimising simple XPath expressions. In *WebDB*, pages 13–18, 2001.

[37] W. Xu and Z. M. Özsoyoglu. Rewriting XPath queries using materialized views. In *International Conference on Very Large Data Bases (VLDB)*, pages 121–132, 2005.

[38] M. Yannakakis. Algorithms for acyclic database schemes. In *International Conference on Very Large Data Bases (VLDB)*, pages 82–94, 1981.