Minimization of Tree Patterns

WOJCIECH CZERWIŃSKI, University of Warsaw, Poland
WIM MARTENS, University of Bayreuth, Germany
MATTHIAS NIEWERTH, University of Bayreuth, Germany
PAWEŁ PARYS, University of Warsaw, Poland

Many of today's graph query languages are based on graph pattern matching. We investigate optimization of tree-shaped patterns that have transitive closure operators. Such patterns do not only appear in the context of graph databases but were originally studied for querying tree-structured data, where they can perform child-, descendant-, node label-, and wildcard-tests.

The *minimization* problem aims at reducing the number of nodes in patterns and goes back to the early 2000's. We provide an example showing that, in contrast to earlier claims, tree patterns cannot be minimized by deleting nodes only. The example resolves the $\mathrm{M} \stackrel{?}{=} \mathrm{NR}$ problem, which asks if a tree pattern is minimal if and only if it is nonredundant. The example can be adapted to prove that minimization is Σ_2^P-complete, which resolves another question that was open since the early research on the problem. The latter result shows that, unless $\mathrm{NP} = \Pi_2^P$, more general approaches for minimizing tree patterns are also bound to fail in general.

CCS Concepts: • **Information systems → Query languages**; • **Theory of computation → Design and analysis of algorithms**;

Additional Key Words and Phrases: XPath, XML, trees, tree patterns, graphs, graph databases, pattern matching, optimization, complexity

ACM Reference Format:
Wojciech Czerwiński, Wim Martens, Matthias Niewerth, and Paweł Parys. 2018. Minimization of Tree Patterns. *J. ACM* 1, 1, Article 1 (January 2018), 45 pages. https://doi.org/10.1145/3180281

1 INTRODUCTION

Tree patterns are a natural and user-friendly means to query graph- and tree-structured data. This is why they can be found in the conceptual core of widely used query languages for graphs and trees.

1.1 Motivation from Graph Query Languages

Graph pattern matching is a fundamental concept in modern declarative graph query languages. Indeed, graph query languages usually take one of two main perspectives: *graph traversal* or *graph pattern matching*, the former being the imperative and the latter being the declarative variant [36]. Today's most prominent declarative graph query languages are SPARQL 1.1 [40] and Neo4J Cypher [31]. Both languages have graph pattern matching at their core: the SPARQL 1.1 specification explicitly states "SPARQL is based around graph pattern matching" [40, Section

Authors' addresses: Wojciech Czerwiński, University of Warsaw, Warsaw, Poland, ; Wim Martens, University of Bayreuth, Angewandte Informatik VII, Bayreuth, Germany; Matthias Niewerth, University of Bayreuth, Angewandte Informatik VII, Bayreuth, Germany; Paweł Parys, University of Warsaw, Warsaw, Poland, .

0004-5411/2018/1-ART1 $15.00
https://doi.org/10.1145/3180281

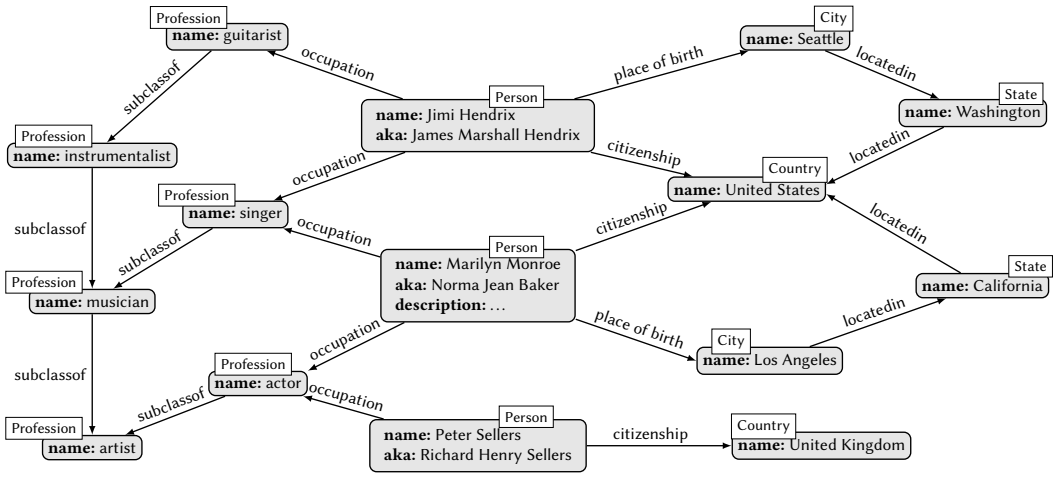


Fig. 1. A graph database (as a *property graph*), inspired by a fragment of WikiData

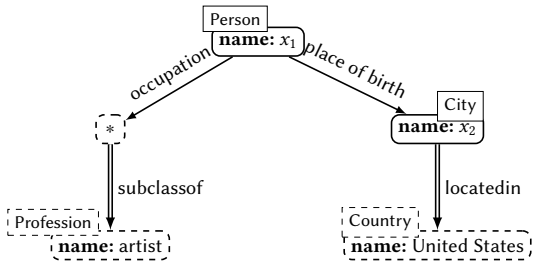


Fig. 2. A tree pattern finding the artists who were born in the United States. The query returns the person names and the cities where they were born. (Fully circled nodes are return nodes.)

5] and the introduction of Neo4J's documentation on Cypher [31, Section 3.1.1] is essentially an introduction to the principles of graph pattern matching. Gremlin [24], another popular graph query language, leans more towards the graph traversal side of the spectrum, but also supports pattern matching style querying. It performs graph pattern matching similar to SPARQL [36].

The reason why graph pattern matching is so popular is not surprising. Graph patterns are expressive, reasonably simple and intuitive to understand, and often efficient to evaluate. Consider the graph in Figure 1. It contains information on artists, their occupation, and their place of birth. The representation is inspired by *property graphs*, a popular model for graph databases in practice [3, 35]. In property graphs, each node and edge carry a label and, in addition, nodes can have a set of attributes. For instance, the node related to Jimi Hendrix has the label Person, its "name" attribute is Jimi Hendrix, and its "aka" attribute is James Marshall Hendrix.

Assume that we would like to find the artists who were born in the United States. This amounts to finding names of Person nodes that have (1) an occupation edge to "a subclass of artist" and (2) a place of birth edge to a city that is located in the United States. For expressing these conditions, we need to reason about paths in the graph. The occupation in (1) should be connected to artist by a path of subclassof-edges and the city in (2) to United States by a path of locatedin-edges.

These conditions are expressed in the pattern in Figure 2.[1] It has two types of edges and two types of nodes. Single edges in the pattern can be matched to single edges in the graph with the same label. The double edges can be matched to *paths* in the graph on which every edge has the label given in the query. (For instance, the locatedin edge in the query can be matched to the path from the Los Angeles node to the United States node.) The solid nodes in the query are *output nodes* and the dashed nodes are ordinary nodes. The symbol "$*$" is a wildcard symbol that can be matched to any label. The query has two variables: x_1 and x_2. Intuitively, computing the answers to the pattern corresponds to finding *matches* of the pattern in the graph and, for each such match, return the nodes (or values) matched by the variables in output nodes of the pattern. When evaluated on the graph in Figure 1, this pattern would return (Jimi Hendrix, Seattle) and (Marilyn Monroe, Los Angeles).

Our example query is structured as a tree. In general, the underlying structure of queries in languages such as SPARQL or Cypher can be an arbitrary graph and can therefore contain cycles. The acyclic queries, however, form an important subclass. Graph patterns closely correspond to *conjunctive queries*, which are known to be NP-complete to evaluate [12]. The tree-shaped patterns closely correspond to *acyclic conjunctive queries*, which can be evaluated in polynomial time. In fact, the quest for subclasses of conjunctive queries with a polynomial time evaluation problem is rich with beautiful results (see, e.g., [22]). In this paper, however, we focus on queries whose underlying structure is a tree and, for this reason, have tractable (polynomial time) evaluation. (We note that the transitive closure operators we use make no difference in this respect.)

From a graph query language perspective, the tree patterns from this paper correspond to tree-shaped conjunctive queries (or tree-shaped graph patterns, if you will) with transitive closure. Such queries are prominent in graph query languages, as indicated by a recent study on SPARQL query logs [11]. Transitive closure seems to be becoming increasingly popular in graph query languages, even though there have been challenges in the early version of the operator in SPARQL 1.1 [5, 28]. In WikiData's list of *example queries* [41], which help users getting started with the data set, 72 out of 272 queries use transitive closure of a label, which means that the feature is relevant. From a theoretical point of view, several variants of such queries have been investigated by [27], who extended their power with data value comparisons and studied their complexity and expressiveness.

1.2 Motivation from Tree Query Languages

Tree-structured data is among us in many forms, JSON and XML being two examples. The tree pattern queries that we consider were originally introduced to investigate query languages for tree-structured data [29]. They are an abstraction of a fragment of XPath [34] and therefore also appear in XQuery, XSLT, and languages for querying JSON [25]. Indeed, patterns such as the one in Figure 2 can equally well be used for querying tree-structured data. (This is easy to see, since a tree is a special case of a graph.)

Tree pattern queries are also important for many topics in fundamental research on tree-structured data. For instance, they form a basis for conjunctive queries over trees [9, 23], for models of XML with incomplete information [7], and for the closely related pattern-based XML queries [16, 21]. They are used for specifying guards in Active XML systems [1] and for specifying schema mappings in XML data exchange [4, 6].

[1] The pattern we show here is closely related to *graph patterns*, which were identified by [3] as a part of the conceptual core of many of today's graph query languages.

1.3 The Core Problem

We investigate the minimization problem for tree patterns. Optimization of queries has been a main topic of database research ever since the beginning and therefore is very natural to consider for tree patterns as well. Tree pattern query optimization already attracted significant attention in the form of *query containment* [15, 29, 32], *satisfiability* [8], and *minimization* [2, 13, 20, 26, 33, 42].

Almost all this former work on query containment, satisfiability, and minimization exclusively considered tree patterns as a language for querying *tree-structured data*. However, as argued by [29, Section 5.3], many of the results hold just the same if we use tree patterns to query graph-structured data.[2]

We make two remarks about the tree patterns that were introduced by [29] (which we consider in most of our proofs), compared to the pattern in Figure 2. The first is that the tree patterns introduced by Miklau and Suciu cannot express the query in Figure 2, for the simple reason that they cannot express the transitive closure of subclassof or locatedin. We will argue that our results extend to these more expressive queries as well.

The second remark is that we will mostly consider Boolean queries, whereas the query in Figure 2 returns tuples of answers. Again, we will argue that our results also apply for higher-arity queries.[3]

We consider the following problem (formally defined in Section 3).

Tree Pattern Minimization	
Given:	A tree pattern p and $k \in \mathbb{N}$
Question:	Is there a tree pattern q, equivalent to p, such that its size is at most k?

We will see that the main difficulties for this problem are already present in a very restricted set of tree patterns that

- only query *graphs that are node-labeled and are tree-shaped*; and
- over these graphs, only use *labeled node tests*, *wildcard node tests*, the *child relation*, and the *descendant relation*.

These are precisely the tree patterns introduced by [29].

1.4 History of the Problem

Although the patterns we consider here have been widely studied [1, 4, 10, 15, 19, 20, 26, 29, 37, 43], their minimization problem remained elusive for a long time. The most important previous work for their minimization was done by [26] and by Flesca et al. [2003, 2008].

The key challenge was understanding the relationship between *minimality* (M) and *nonredundancy* (NR). 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[4]) can be deleted while remaining equivalent. The question was if minimality and nonredundancy are the same ([26, Section 7] and [20, page 35]):

M $\stackrel{?}{=}$ NR Problem
Is a tree pattern minimal if and only if it is nonredundant?

[2] We discuss this in Section 2.

[3] [26, Section 5.2] proved that containment for k-ary queries can be reduced to the Boolean case. It is not clear whether their reduction also proves that minimization can be reduced.

[4] [26, Proposition 3.3] proved that a tree pattern has a redundant branch if and only if it has a redundant leaf.

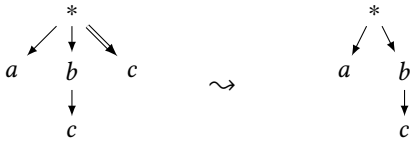


Fig. 3. Minimizing a tree pattern by removing redundant nodes

Notice that a part of the $M \stackrel{?}{=} NR$ problem is easy to see: a minimal pattern is trivially also nonredundant (that is, $M \subseteq NR$). The opposite direction is much less clear.

If $M = NR$, then the simple algorithmic idea summarized in Algorithm 1 would correctly minimize tree patterns. Therefore, the $M \stackrel{?}{=} NR$ problem is a natural question about the design of minimization algorithms for tree patterns.

Algorithm 1 Computing a nonredundant subpattern

Input: A tree pattern p
Output: A nonredundant tree pattern q, equivalent to p

1: **while** a leaf of p can be removed (remaining equivalent to p) **do**
2: Remove the leaf
3: **end while**
4: **return** the resulting pattern

Example 1.1. It is easy to see that Algorithm 1 can be used for minimizing some patterns. Consider the left pattern in Figure 3. Its root (labeled with a wildcard "$*$") can be matched to nodes n in a graph such that (1) n has an a-labeled successor, (2) n has a b-labeled successor with a c-labeled successor, and (3) from n a c-labeled node is reachable. (In this example, edge labels do not matter.) In the semantics of such patterns, it is allowed that the different c-nodes are matched to the same node in the data. Therefore, condition (3) is redundant and the pattern to the right is equivalent and smaller.

The $M \stackrel{?}{=} NR$ problem is also a question about complexity. The main source of complexity of the nonredundancy algorithm lies in testing equivalence between a tree pattern p and a tree pattern obtained from p by removing a leaf on line 1. This test is generally coNP-complete [29]. If $M = NR$, then Tree Pattern Minimization would also be coNP-complete.

In fact, the problem was claimed to be coNP-complete by [19, Theorem 2], but the status of the minimization and the $M \stackrel{?}{=} NR$ problems were re-opened by [26], who found errors in the proofs. Flesca et al.'s journal paper 2008 then proved that $M = NR$ for a limited class of tree patterns, namely those where *every wildcard node has at most one child*. Nevertheless, for tree patterns,

(a) the status of the $M \stackrel{?}{=} NR$ problem and
(b) the complexity of the minimization problem

remained open.

1.5 Our Contributions

We will prove the following:

(a) There exists a tree pattern that is nonredundant but not minimal. Therefore, $M \neq NR$.

(b) Tree Pattern Minimization is Σ_2^P-complete. This implies that even the main idea in Algorithm 1 cannot work unless coNP = Σ_2^P.

Interestingly, our counterexample for (a) uses only two wildcard nodes with two children and only one transitive edge. This is only barely beyond the fragment from [20] for which it is known that minimality and nonredundancy coincide.

Outline. In Section 2 we formally define tree patterns, their semantics, and basic notions that we will use throughout the article. We formally state our main results in Section 3, discuss the relationship between Boolean and k-ary tree patterns, and discuss the relationship between tree patterns and the queries in introduction. We will also see that, for many of our results, it suffices to only consider tree patterns that query trees (instead of graphs). In Section 4, we introduce preliminaries for tree patterns that we only need when we consider them to query trees. We show why M ≠ NR in Section 5 and prove that minimal patterns are not unique in a strong way, that is, they can have different shapes. In Section 6 we present the main reductions for the complexity of the minimization and minimality problems. We postpone the proof of one technical lemma to Section 7. We conclude in Section 8.

This article is based on [14] and extends the work in the following way. In addition to considering tree patterns as a query language for tree-structured data, we now also define tree patterns as a query for graph-structured data and prove our results in both contexts. Moreover, we provide the results for *generalized* tree patterns, which have labeled transitive closure edges, which were not present in [14]. In addition, we provide detailed proofs for the claims made in [14].

2 PRELIMINARIES

We formally define our data model, our queries, and recall important results about the static analysis of queries.

2.1 Data Model: Node- and Edge-Labeled Graphs and Trees

Our data models are finite, node- and edge-labeled graphs and trees. The labels can come from a countably infinite set Λ.[5] In the graph database world, our model is closely related to *property graphs*, the data model for Neo4J [35] (see, e.g., [3] for a formal definition of property graphs).[6]

More formally, a *(node- and edge-) labeled graph* is a triple

$$G = (V, E, \text{lab}),$$

where V is a finite nonempty set of *nodes*, E is a set of directed *edges* $(u, v) \in V \times V$ and $\text{lab}\colon V \cup E \to \Lambda$ is a *labeling function* assigning to every node and edge its label coming from an infinite set of labels Λ. In this article, we always assume that graphs are connected. A connected graph is a *tree* if,

(i) for every node v, there is at most one node u (called *parent* of v) with $(u, v) \in E$ and
(ii) there is exactly one node v (called *root*) without a parent.

For trees, we refer to E as the *child* relation and E^{-1} as the *parent* relation. The *descendant* and *ancestor* relations are transitive closures of the child and parent relations, respectively.

[5] We choose the set of labels to be infinite because, in real-world data models such as RDF or XML, users can use strings to label nodes. Intuitively, our labels correspond to such strings. The infinite size of the set of labels is also needed for some of the techniques we use, see Section 3.4.

[6] Property graphs are more refined, however, since they associate *properties* to nodes in addition to labels. From a formal perspective, we want that nodes in the graph are not uniquely determined by their label. In particular, we do not want that different occurrences of a label in a query must always be mapped to the same node in the graph. This behavior would introduce unwanted cycles in tree pattern queries.

A *path* of length n from a node v_0 to a node v_n is a sequence of nodes

$$\pi = v_0 \cdots v_n$$

where $(v_{i-1}, v_i) \in E$ for every $i = 1, \ldots, n$.

2.2 The Queries: (Generalized) Tree Patterns

We will consider two kinds of tree patterns. The first kind has only node labels and we just call them *tree patterns* (which is consistent with the literature [15, 20, 26, 29]). The second has node- and edge-labels and we will call them *generalized tree patterns*.

More precisely, our formal model of patterns allows node- and edge-label tests, wildcard tests, and transitive closures. The wildcard test (denoted by "$*$" in patterns) matches any node- or edge-label in a graph. To avoid confusion, we assume that $* \notin \Lambda$.

A *generalized tree pattern* is a tuple

$$p = (V_p, E_p, \mathrm{lab}_p)$$

where

(1) $\mathrm{lab}_p \colon V_p \cup E_p \to \Lambda \uplus \{*\}$,
(2) $(V_p, E_p, \mathrm{lab}_p)$ satisfies the conditions (i) and (ii) we required for trees, and
(3) E_p is partitioned in two sets: *simple edges* and *transitive-closure edges*.

If a node (resp., edge) is labeled "$*$", we call it a *wildcard node* (resp., *wildcard edge*). We will regularly represent generalized tree patterns graphically. When we do so, we draw transitive-closure edges using double lines. Furthermore, we omit wildcard symbols on edges for readability and for consistency with the literature on tree patterns.

We say that a generalized tree pattern is just a *tree pattern* if all its edges are wildcard edges. The *size* of a (generalized) tree pattern p, denoted $\mathrm{size}(p)$, is the number of its nodes.

For simplicity, we will define our queries to be Boolean, that is, we will only consider whether they can be matched in a graph or not. Tree patterns with output nodes have been considered as well [26, 29] and our main results also apply to those queries. We discuss this in Section 3.3.

2.3 Semantics of Queries

The semantics for patterns is based on homomorphisms. Intuitively, a pattern can be matched in a graph if there exists a mapping from the pattern to the graph that satisfies all constraints imposed by the pattern.

More precisely, for a generalized tree pattern $p = (V_p, E_p, \mathrm{lab}_p)$ and a labeled graph $G = (V, E, \mathrm{lab})$, a function $\mathrm{m} \colon V_p \to V$ is a *match* of p in G if it fulfills all the following conditions:

(1) If $\mathrm{lab}_p(v) \neq *$ for $v \in V_p$ then $\mathrm{lab}_p(v) = \mathrm{lab}(\mathrm{m}(v))$.
(2) If $(u, v) \in E_p$ is a simple edge then $(\mathrm{m}(u), \mathrm{m}(v))$ is an edge in G. Furthermore, if $\mathrm{lab}_p((u, v)) \neq *$ then $\mathrm{lab}_p((u, v)) = \mathrm{lab}((\mathrm{m}(u), \mathrm{m}(v)))$.
(3) If $(u, v) \in E_p$ is a transitive-closure edge then there is a path of positive length from $\mathrm{m}(u)$ to $\mathrm{m}(v)$ in G that satisfies the label constraint of the edge. That is, there exists a path $\pi = u_0 \cdots u_n$ in G such that $\mathrm{m}(u) = u_0$, and $\mathrm{m}(v) = u_n$, and $n > 0$. Moreover, if $\mathrm{lab}_p((u, v)) \neq *$, then all edges (u_{i-1}, u_i) in π are labeled with $\mathrm{lab}_p((u, v))$.

We say that p *can be matched in* G if there exists a match from p to G. Figure 4 shows an example of a generalized tree pattern, a labeled graph, and a match. Notice that we do not require matches to be injective.

Definition 2.1 (Semantics of Generalized Tree Patterns). The *set of models of a (generalized) tree pattern* p, denoted by $M(p)$, is the set of graphs in which p can be matched. The *set of tree models of*

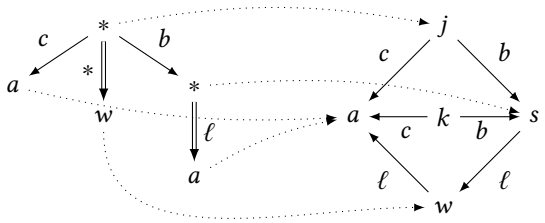


Fig. 4. Example of a match of a generalized tree pattern (left) in a labeled graph (right)

Fig. 5. Example for containment of patterns. (Non-labeled edges are implicitly assumed to have wildcard tests.)

p, denoted by $M_t(p)$, is the subset of $M(p)$ containing only trees, that is, the set of trees in which p can be matched.

Terminology for Tree Models. In a significant part of the article (Sections 4–7) we will only consider tree models of patterns. In this context, if (u, v) is a simple edge in pattern p, then $\mathrm{m}(v)$ is always a child of $\mathrm{m}(u)$ in G. Likewise, if (u, v) is a transitive-closure edge, then $\mathrm{m}(v)$ is always a descendant of $\mathrm{m}(u)$. Therefore, when we only consider tree models, we will use the terms *child edges* and *descendant edges* to refer to simple edges and transitive-closure edges, respectively.

2.4 Equivalence and Containment

Many optimization algorithms use equivalence or containment tests as subprocedures. In this paper, equivalence and containment will therefore play a central role. We define these notions next.

Definition 2.2 (Containment, Equivalence). Let p_1 and p_2 be generalized tree patterns.

- We say that p_1 is *contained* in p_2 if $M(p_1) \subseteq M(p_2)$, which we denote by $p_1 \subseteq p_2$.
- We say that p_1 is *equivalent* to p_2 if $M(p_1) = M(p_2)$, which we denote by $p_1 \equiv p_2$.

Notice that $p_1 \equiv p_2$ if and only if $p_1 \subseteq p_2$ and $p_2 \subseteq p_1$.

Example 2.3. Figure 3 contains two patterns that are equivalent. (For the left pattern, the c-labeled node on the right branch can always be matched to wherever the c-labeled node in the middle branch is matched. Therefore it is equivalent to the pattern on the right.) In Figure 5, we give an example for pattern containment. The right pattern matches in graphs that have an a-node which has c_1- and c_2-nodes on distance two, such that there are b-nodes between the a and the c_i. The pattern on the left additionally requires the two b-nodes to be the same. Therefore, if the pattern on the left can be matched, the pattern on the right can also be matched.

The following decision problems are central in many query optimization procedures:

(Generalized) Tree Pattern Equivalence	
Given:	Two (generalized) tree patterns p_1 and p_2
Question:	Is $p_1 \equiv p_2$?

(Generalized) Tree Pattern Containment	
Given:	Two (generalized) tree patterns p_1 and p_2
Question:	Is $p_1 \subseteq p_2$?

We consider these problems for generalized tree patterns as well as for ordinary tree patterns. The following proposition shows that it suffices to consider tree models to decide these containment and equivalence problems.

Proposition 2.4. *Let p_1 and p_2 be generalized tree patterns. Then*

$$p_1 \subseteq p_2 \text{ if and only if } M_t(p_1) \subseteq M_t(p_2) \ .$$

The same holds if p_1 and p_2 are tree patterns.

Sketch. If p_1 and p_2 are tree patterns, the result is already known [29, Section 5.3]. The proof of Miklau and Suciu is by unfolding of graph models to trees. Their proof can be directly applied to our setting. The only difference is that, in our setting, the trees and graphs can carry edge labels. For completeness, we provide the minor adaptation of Miklau and Suciu's proof in Appendix A. □

Leveraging Proposition 2.4, [29, Theorem 4] proved that Tree Pattern Equivalence and Tree Pattern Containment are coNP-complete. The results are not difficult to extend to generalized tree patterns. (A proof can be found in Appendix A.)

Proposition 2.5. *Generalized Tree Pattern Containment is* coNP*-complete.*

From the coNP upper bound in Proposition 2.5 and the coNP lower bound for tree patterns [29, Theorem 4], we can immediately infer the following.

Corollary 2.6. *Generalized Tree Pattern Equivalence is* coNP*-complete.*

2.5 Minimality and Nonredundancy

We call a generalized tree pattern p *redundant* if one of its nodes can be removed without changing its set of models. For a node v of p, we denote by $p \setminus v$ the pattern obtained from p by removing v and all its descendants and incident edges.

Definition 2.7 (Minimality, Nonredundancy). Let p be a generalized tree pattern.

- We say that p is *redundant* if it is 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*.
- We say that p is *minimal* if there exists no generalized tree pattern that is equivalent to p but has strictly smaller size.

Recall the definition of the M $\stackrel{?}{=}$ NR problem from the introduction, asking if minimality and nonredundancy is the same. [20] studied this problem for tree patterns (so, every edge has a wildcard) and identified the following important class.

Definition 2.8 ($*$-narrow tree pattern). A tree pattern is a *$*$-narrow tree pattern* if every wildcard node has at most one child.

Lemma 2.9 ([20, Corollary 4.15]). *Let p be a $*$-narrow tree pattern. Then p is minimal if and only if it is nonredundant.*

3 MAIN RESULTS

We can now formally define the main decision problem that we study in this article:

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

Again, there are two versions of this problem. The generalized version takes a generalized tree pattern as input and searches for small equivalent generalized tree patterns. The other version of the problem only considers ordinary tree patterns.

Furthermore, we can formally state our main results:

(M1) There exists a tree pattern that is nonredundant but not minimal (Section 5.1).
(M2) Minimal tree patterns are not unique (Section 5.2). The non-uniqueness holds in a strong sense that we clarify in Section 5.2.
(M3) Generalized Tree Pattern Minimization and Tree Pattern Minimization are Σ_2^P-complete. We prove the upper bound for Generalized Tree Pattern Minimization in Section 3.1. The lower bound already holds in two special cases that are interesting in their own right:
- Tree Pattern Minimization is Σ_2^P-hard. This case is interesting in the context of tree-structured data, since the tree patterns are a fragment of XPath. We will prove it in Sections 6 and 7.
- Generalized Tree Pattern Minimization is Σ_2^P-hard even if wildcard edges are not used. This case is interesting in the context of graph databases. (See, e.g., the example in the introduction, where we use labels on every edge.) This case also follows from our main proof in Sections 6 and 7, see Corollary 6.8.

3.1 Main Complexity Results

We will prove the following:

Theorem 3.1. *Tree Pattern Minimization is Σ_2^P-complete.*

Theorem 3.2. *Generalized Tree Pattern Minimization is Σ_2^P-complete.*

The upper bound for both Theorems is immediate from Lemma 3.3, which we prove here. The lower bounds of Theorems 3.1 and 3.2 follow from Lemma 6.1 and Corollary 6.8, respectively, which we prove in Section 6.

Lemma 3.3. *The following two problems are in Σ_2^P.*

(1) Generalized Tree Pattern Minimization
(2) Tree Pattern Minimization

Proof. We first prove (1). Given a generalized tree pattern p and $k \in \mathbb{N}$, the Σ_2^P algorithm first guesses (existentially) a generalized tree pattern q of size at most $\min(k, |p|)$ and then checks (universally) if p and q are equivalent. Clearly q is polynomially large. Therefore, due to Corollary 2.6, the universal phase also takes a polynomial number of steps.

The proof for (2) is completely analogous. It just considers tree patterns instead of generalized tree patterns everywhere. □

We note that these results extend to *forests*, i.e., conjunctions of tree patterns. Indeed, for tree patterns $p_1, \ldots, p_n$, the query $p_1 \wedge \cdots \wedge p_n$ (with the obvious semantics) can be minimized by the following algorithm:

1. Construct the tree pattern p consisting of a new wildcard root, with transitive-closure edges to the root of each p_i.
2. Minimize p.
3. Remove the root of p (yielding again a forest).

The resulting forest might have less than n tree patterns, in the case that some pattern was redundant. We note that p is not necessarily exactly equivalent to the conjunction, as p requires an additional node above some embedding of the conjunction. However, p is equivalent to the conjunction $p'_1 \wedge \cdots \wedge p'_n$, where p'_i results from p_i by adding a new root and connecting it with a transitive-closure edge to the root of p_i.

3.2 Non-Uniqueness Result

We provide some context for the non-uniqueness result (M2), since it contrasts a well-known result for conjunctive queries. It is well known that minimal conjunctive queries are unique up to isomorphisms (i.e., up to renaming of variables) [12]. This minimal query can be computed by computing the so-called *core* of its associated hypergraph. For conjunctive queries, however, minimization is NP-complete.

In Section 5.2, we do not only show that minimal tree patterns are not unique up to isomorphisms, but we also show that their tree structure can be different. That is, even disregarding the difference between simple edges and transitive-closure edges, the relation E_p of edges can be different.

3.3 Boolean Versus k-ary Queries

One can easily extend tree patterns to k-ary tree patterns that return k-tuples of answers (see, e.g., [26, 29]). We argue that our main results (M1–M3) also apply to such queries. Concerning (M1) and (M2), this is trivial. Our examples showing that minimality is different from nonredundancy (Section 5.1) and that minimal queries are not unique (Section 5.2) are Boolean queries, which are special cases of k-ary queries ($k = 0$).

Concerning (M3), our main complexity results are the Σ_2^P-completeness results in Theorems 3.2 and 3.1. For Tree Pattern Minimization, the Σ_2^P upper bounds can be seen to hold for k-ary queries by using the same naive algorithm as in Lemma 3.3 and using the following argument of [26, Section 5.2] for showing that Tree Pattern Equivalence for k-ary queries polynomially reduces to the same problem for Boolean queries. A tree pattern with k output nodes $(o_1, \ldots, o_k)$ is converted to a Boolean tree pattern by

(1) attaching to each output node o_i a new child edge to a new node with label τ_i (here, τ_i is a new label that we assume not to appear elsewhere in the query) and
(2) attaching to each leaf node not in $\{o_1, \ldots, o_k\}$ a new child edge to a new wildcard node.

Then, [26, Proposition 5.2] prove that, for two k-ary queries p and q, we have that p is contained[7] in q if and only if the Boolean version of p is contained in the Boolean version of q.

For Generalized Tree Pattern Minimization, we can use precisely the same argument. The only difference in the construction is that we need to add edge labels. But edge labels are of no importance in the proof. Kimelfeld and Sagiv's proof goes through if we label all newly created child edges with wildcards, and also if we would label all of them with the same label. (This label can even appear elsewhere in the query.) For reducing to containment of two queries p and q, it

[7]Here, containment means that, for every tree t every k-tuple selected by p is also selected by q.

is important that we choose the same label for both queries. The Σ_2^P lower bound is immediate because we prove it for $k = 0$.

3.4 Finite Versus Infinite Domains and Schema Information

In Section 2, we define our graphs and trees such that their labels come from an infinite set Λ. This assumption is important for some of the techniques we use. For instance, the definition of canonical tree models in Section 4.2 requires a label z that does not occur in any of the tree patterns we consider in the paper. This construction and the proof of Lemma 4.1 rely on the infinity of Λ. That said, it is easy to see that some of our results still hold if Λ is finite. For instance, our counterexample for (M1) is still valid if $|\Lambda| = 5$, since the patterns use four labels.

The discussion on the finiteness of Λ is closely connected to minimization of tree patterns *with schema information*. This question has been extensively studied in the literature on query containment for tree-structured data, see, e.g., [8, 10, 15, 32, 42]. It is well-known that the complexity of minimization increases if schema information is present. For instance, even the *validity problem* is EXPTIME-complete with respect to schema information [10, 15]. More precisely, if S is a schema defining a set of trees $T(S)$ and p is a tree pattern, then it is EXPTIME-complete to decide if p matches all trees in $T(S)$, that is $T(S) \subseteq M(p)$. (Notice that, if $T(S) \subseteq M(p)$, then *on the trees* $T(S)$, pattern p is equivalent to a trivial pattern '$*$' that matches every non-empty tree.) The EXPTIME lower bound even holds if p does not branch and S is given as a tree automaton or a formalism for XML Schema [10].

Shedding more light on (generalized) tree pattern minimization for finite alphabets or in the context of schema information would be a valuable extension of our work.

3.5 Relationship to the Queries in the Introduction

The tree patterns we defined here are much simpler than the pattern we discussed in the introduction (Figure 2). However, the two types of patterns are closely related when it comes to minimization. Again, since the patterns we have here are simpler, it is easy to see that our M $\neq$ NR example equally applies to the kind of patterns in the introduction.

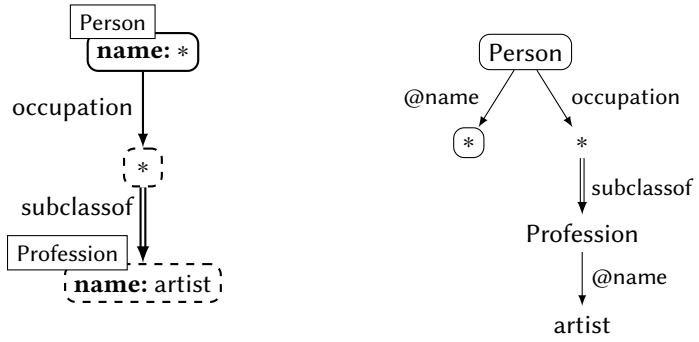
Moreover, the simplified patterns capture much of the expressivity of the more complex patterns modulo a simple encoding. In Figure 6, we demonstrate this translation by example, using a subquery of Figure 2. Here, each node of the pattern on the left becomes a node on the right labeled with the *property* (the label in the rectangular box) if present, and the "name"-attributes of nodes become children with incoming edges that identify the type of attribute. (We can make sure that the labels of these incoming edges do not appear elsewhere in the query.)

The example shows that there is a very close correspondence between property graph patterns and the generalized tree patterns we consider. Of course, for the minimization of property graph patterns, one may be interested in minimizing a different cost function as the one obtained by taking the size of the translated generalized tree pattern. Nevertheless, for many such cost functions, it will be possible to translate our results to the property graph setting using a simple encoding as the one in Figure 6.

4 PRELIMINARIES FOR TREE PATTERNS

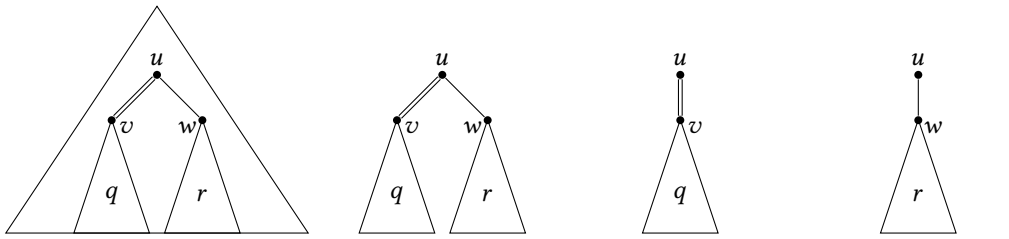
All the results from here on in the article can be proved by only considering tree patterns. Therefore, we will no longer consider generalized tree patterns in the remainder, except for Corollary 6.8. Here we define technical notions for tree patterns that we need for our proofs.

Furthermore, due to Proposition 2.4, we will be able to almost exclusively consider tree models from here on. (We make it explicit where we consider graphs.) We will therefore simplify our notation in figures and omit the arrows on edges. All edges are assumed to be pointed downwards.



(a) Property graph pattern (b) Translated generalized tree pattern

Fig. 6. Translating a subquery of Figure 2 to our simplified model



(a) A pattern p (b) Subpattern p^u (c) Subpattern p^u_v (d) Subpattern p^u_w

Fig. 7. Notation for subtrees and subpatterns. Labels and wildcards are inherited from p.

4.1 Basics

For a tree pattern p and node u, the *depth* of u in p is the number of edges on the path from the root of p to u. For a tree pattern p and node u we use p^u to denote the subpattern of p rooted at u, that is, the tree pattern obtained from p by restricting its set of nodes to u and all its descendants (and by restricting the edges accordingly). For a child v of the root of p, we denote by p_v the pattern consisting of the root of p, connected to the subpattern p^v in the same way as they are connected in p. We illustrate the notation in Figure 7. In Figure 7a we assume that u has precisely two children. Then, the root of p^u also has exactly two children (Figure 7b) and the types of edges are inherited from p. Figures 7c and Figures 7d illustrate the notation with subindices but already start from the pattern p^u. We use the same notation for trees.

4.2 Canonical Tree Models

Canonical tree models were introduced by [29] and are special tree models of patterns. They are structured similarly as the pattern and they are interesting because they simplify the containment problem: a tree pattern p is contained in a tree pattern q if and only if all canonical tree models of p are matched by q. We make extensive use of canonical tree models in our proofs.

Let $z \in \Lambda$ 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 Λ is infinite.) A *canonical tree model* of a tree pattern $p = (V, E, \mathrm{lab}_p)$ is a tree $t = (V_t, E_t, \mathrm{lab}_t)$ obtained from p by application of the two following steps:

- for every node v such that $\mathrm{lab}_p(v) = *$, define $\mathrm{lab}_t(v) = z$,

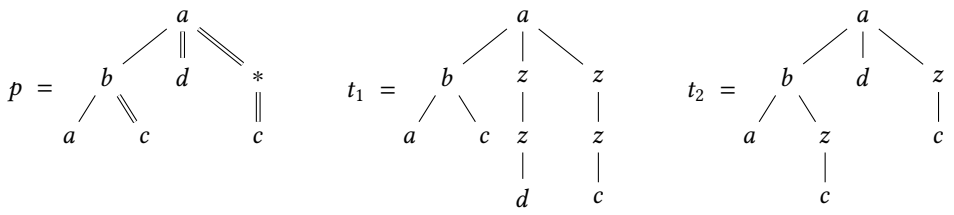


Fig. 8. A tree pattern p and two canonical tree models t_1 and t_2

- 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 we always have that $V \subseteq V_t$. Furthermore, it is possible that in the last step no new nodes are created, in which case $V = V_t$. This happens when each descendant edge is replaced by a single child edge. An example of a tree pattern p and two of its canonical tree models is given in Figure 8. We denote by $\mathrm{Can}(p)$ the set of all canonical tree models of p.

The following lemma is very similar to Proposition 3 in [29]. The difference is that matches in [29] additionally require that the root of the pattern is matched to the root of the tree. For completeness we give a proof that works using our definition of matches.

Lemma 4.1. *Let p and q be tree patterns. Then $p \subseteq q$ if and only if $\mathrm{Can}(p) \subseteq M_t(q)$.*

Proof. The "only if" direction is immediate, because $\mathrm{Can}(p)$ is contained in $M_t(p)$, which is, by Proposition 2.4, contained in $M_t(q)$.

In order to show the "if" direction, we show the contraposition, i.e., we show $p \not\subseteq q$ implies $\mathrm{Can}(p) \not\subseteq M_t(q)$. From $p \not\subseteq q$ we can conclude by Proposition 2.4 that $M_t(p) \not\subseteq M_t(q)$. Therefore, there exists a tree $t_0 \in M_t(p)$ such that $t_0 \notin M_t(q)$. We will apply some modifications to the tree t_0 obtaining a tree in $\mathrm{Can}(p)$, but not belonging to $M_t(q)$. Let V_p be the set of nodes of p, let V be the set of nodes of t_0, and let $\mathrm{m}\colon V_p \to V$ be a match of p in t_0. Then let t_1 be the tree obtained from t_0 by removing all the nodes that do not have a descendant in $\mathrm{m}(V_p)$, and all the nodes that do not have an ancestor in $\mathrm{m}(V_p)$. Observe that p also matches in t_1, by the same match m. We also still have $t_1 \notin M_t(q)$. Let t_2 be the tree t_1 relabeled appropriately, that is, every node of t_1 which is not equal to $\mathrm{m}(v)$ for some non-wildcard node v is relabeled to z. Note that still $t_2 \in M_t(p)$ and $t_2 \notin M_t(q)$. Moreover, t_2 is now a canonical tree model of p. Thus indeed $\mathrm{Can}(p) \not\subseteq M_t(q)$, which finishes the proof. □

4.3 Canonical Matches

Given a pattern p and one of its canonical tree models t, there is a straightforward (injective) match of the non-wildcard nodes of p into t. More precisely, since $V \subseteq V_t$, this match is the identity on V. We sometimes use this correspondence to reason about nodes in t. That is, for a non-wildcard node u of p, we use this injective match to identify *the node in t corresponding to u*. (Sometimes, in order to shorten the presentation, we even identify these nodes with each other and use u to refer to the node in p as well as the corresponding node in t. Which node we mean will always be clear from the context.)

We sometimes also consider injective matches m_t of p in t such that, for every node u of p, we have that $t^{\mathrm{m}_t(u)}$ is a canonical tree model of p^u. We call such a match a *canonical match*. Notice that the straightforward (injective) match of p in t we mentioned before fulfills this property, so canonical matches always exist between tree patterns and their canonical tree models. On the other

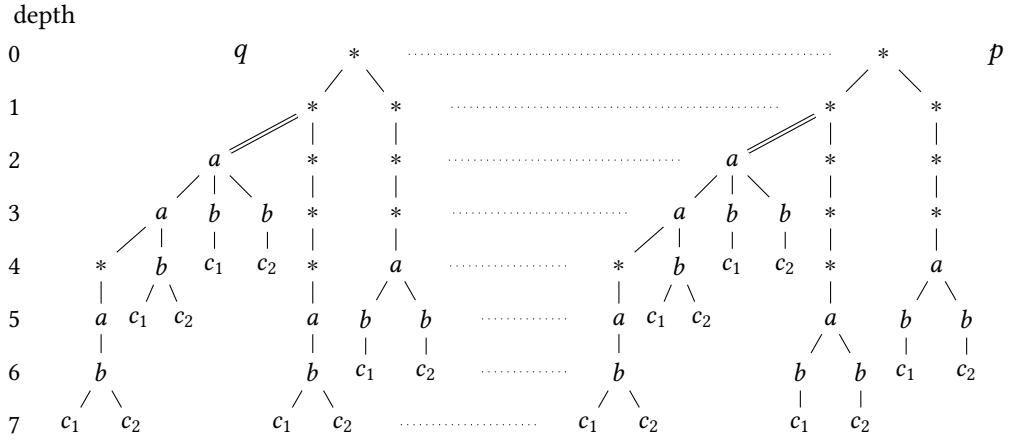


Fig. 9. A nonredundant tree pattern p (right) and an equivalent tree pattern q that is smaller (left)

hand, given a tree pattern p and its canonical tree model t, there may be more than one canonical match.

For example, the pattern p from Figure 8 has two canonical matches in the canonical tree model t_1: one where the wildcard node maps to the rightmost child of the root and one where it maps one level deeper.

4.4 Homomorphisms Between Tree Patterns

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

(1) if $\text{lab}_{p_1}(v) \neq *$ for $v \in V_{p_1}$ then $\text{lab}_{p_1}(v) = \text{lab}_{p_2}(\text{h}(v))$,
(2) if $(u, v) \in E_{p_1}$ is a child edge then $(\text{h}(u), \text{h}(v)) \in E_{p_2}$ is a child edge, and
(3) if $(u, v) \in E_{p_1}$ is a descendant edge then $\text{h}(u)$ is a proper ancestor of $\text{h}(v)$ in p_2.

The existence of a homomorphism from p_1 to p_2 is a sufficient condition for $p_2 \subseteq p_1$ [29].

Observation 4.2. *If there is a homomorphism from tree pattern p_1 to tree pattern p_2, then $p_1 \subseteq p_2$.*

The reason is that, if there exists a match m of p_2 in a tree t, then $\text{m} \circ \text{h}$ is a match of p_1 in t. We make use of this fact later in the article.

5 NONREDUNDANCY AND MINIMALITY

In this section, we resolve the $\text{M} \stackrel{?}{=} \text{NR}$ problem by presenting a tree pattern that is nonredundant but also not minimal. 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 6.

5.1 Nonredundancy ≠ Minimality

To prove that nonredundancy is different from minimality, we will argue that the right pattern p in Figure 9 is nonredundant and not minimal. Consider the pattern q on the left of Figure 9. We need to make three points: (1) p is nonredundant, (2) p is equivalent to q, and (3) q is smaller than p.

Point (3) is trivial: q can be obtained from p by merging two b-nodes on depth six. Therefore, q has one node fewer than p. Points (1) and (2) are non-trivial and we show them next.

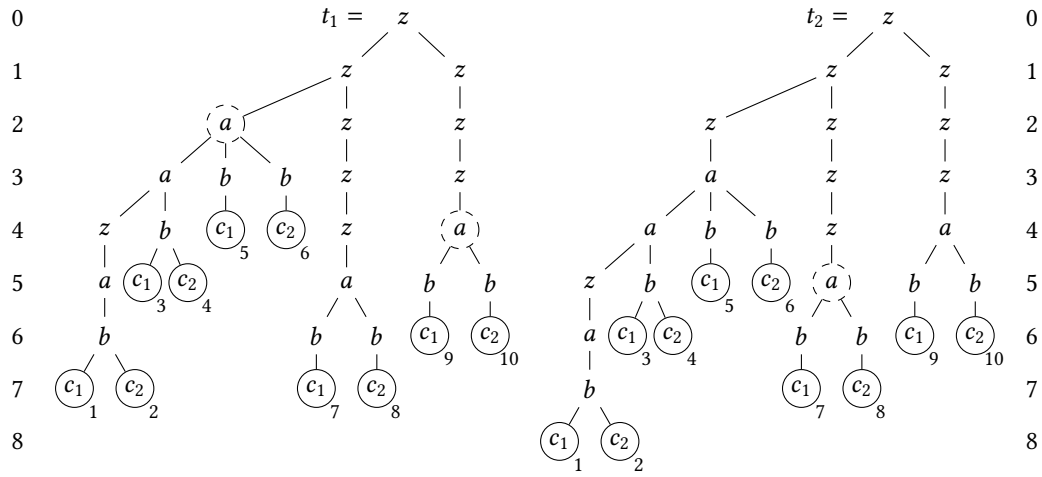


Fig. 10. Trees for proving nonredundancy of p in Figure 9

Pattern p is nonredundant. We will use the following proposition from Kimelfeld and Sagiv that allows us to simplify the proof.[8]

Proposition 5.1 ([26, Proposition 3.3]). *A tree pattern is redundant if and only if it has a redundant leaf.*

In other words, it suffices to show that none of the leaves of p can be deleted while remaining equivalent. For the purpose of this section, we order the leaves in p from left to right, that is, the *first leaf* is the leftmost c_1-leaf on depth 7, the *fourth leaf* is the c_2-leaf on depth 5, etc.

Figure 10 contains two canonical tree models of p. If an arbitrary leaf n of p is removed, then the resulting pattern $p \setminus n$ always matches the trees $t_1 \setminus n$ and $t_2 \setminus n$.[9] We will show that, no matter which leaf n we choose, either $t_1 \setminus n$ or $t_2 \setminus n$ will no longer be matched by the original pattern p. This proves that, for every leaf n, pattern $p \setminus n$ is not equivalent to p.

Let thus n be one of the leaves of p. We denote by t_k this tree among t_1, t_2 in which n is circled in Figure 10. We will prove that p cannot be matched in $t_k \setminus n$. Let v be the dashed a-node above n in t_k (and, simultaneously, the corresponding node of p). In each of the ten cases v cannot be matched to itself, as then n should be matched to some c_1-node (resp. c_2-node) being below v and on the same depth as n, but there is no such node in $t_k \setminus n$. It cannot be also matched to any node having smaller depth, because the depth of v in p and in t_k is the same, and above the image of v there should be enough place to match the path from v up to the root. As there is no other a-node on the same depth as v, this means that v has to be matched to some node being deeper in the tree. This causes a problem:

- If n is one of the six leftmost leaves, we notice that in $t_1 \setminus n$ there is no a-node with an a-child on depth 3 or greater.
- Suppose that n is the seventh or the eighth leaf. Then v has to be matched to the a-node on depth 6. It follows that the rightmost a-node of p has to be matched one level higher. The

[8] They proved it for tree patterns that only accept tree models, but due to Proposition 2.4, their result also applies if we consider graph models.

[9] We use here that the nodes of p are a subset of the nodes of t_1 and t_2; see Section 4.2.

only a-node on depth 5 in $t_2 \setminus n$ is v. Below v, however, the node n is missing, which makes it impossible to match there.
- Finally, suppose that n is one of the two rightmost leaves. In this case, if v is matched on depth greater than 4, it will be impossible to match the seventh leaf of p, because the tree $t_1 \setminus n$ is too shallow.

This shows that pattern p is nonredundant.

Pattern p is not minimal. Finally, we show that the tree pattern p is equivalent to the tree pattern q. To this end, observe that $q \subseteq p$ because q is more restrictive: it has the same requirements as p but, in addition, it says that the nodes to which the seventh and the eighth leaves are matched have the same parent. More formally, it is also easy to see that there is a homomorphism from p to q. It therefore only remains to show that $p \subseteq q$. By Lemma 4.1, it suffices to prove that $\mathrm{Can}(p) \subseteq M_t(q)$.

In Figure 11, we depicted q (always on the left) and the three possible cases t_1, t_2 and $t_{\geq 3}$ of canonical tree models of p on the right. Since p has only one descendant edge, its canonical tree models only differ in the number of edges that are introduced to replace this descendant edge. We consider five cases, depending on whether the descendant edge is replaced by

(a) one edge (t_1),
(b) two edges (t_2), or
(c) three edges (t_3), or
(d) four edges (t_4), or
(e) at least five edges ($t_{\geq 5}$).

These five possibilities are depicted on the right of Figure 11. In all cases, the dotted arrows show how q can be matched in the respective tree. (The gray parts of the canonical trees are parts that are not needed for matching q.)

We therefore obtained the following:

Theorem 5.2 (M $\neq$ NR). *There exists a tree pattern that is nonredundant and not minimal.*

5.2 Minimal Patterns are not Unique

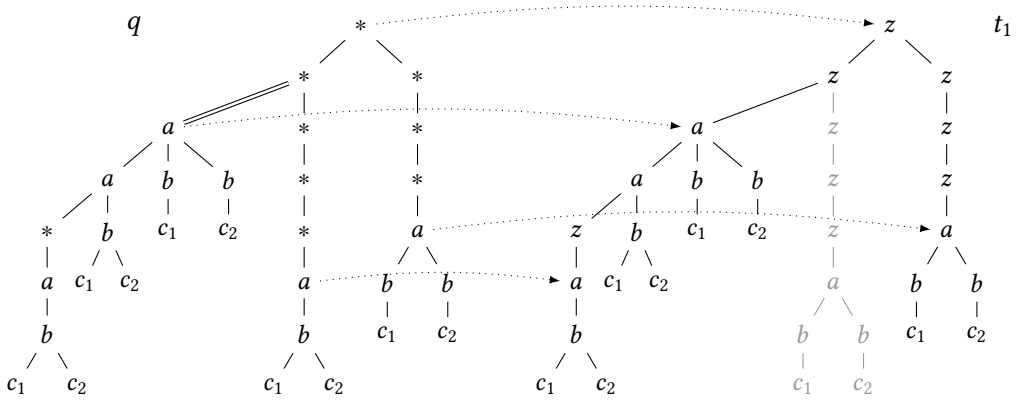
It is easy to see that minimal tree patterns are not unique. For example, the tree patterns in Figure 12a are different, minimal, and both express that there should be at least one node between an a-labeled node and a b-labeled node.

It can be argued, however, that the difference between the patterns in Figure 12a 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 [29, 30]. On the other hand, [29, Section 3.2] proved that, in restricted cases, it is possible to rewrite patterns in a normal form (which they call *adorned tree patterns*) that alleviates this problem.

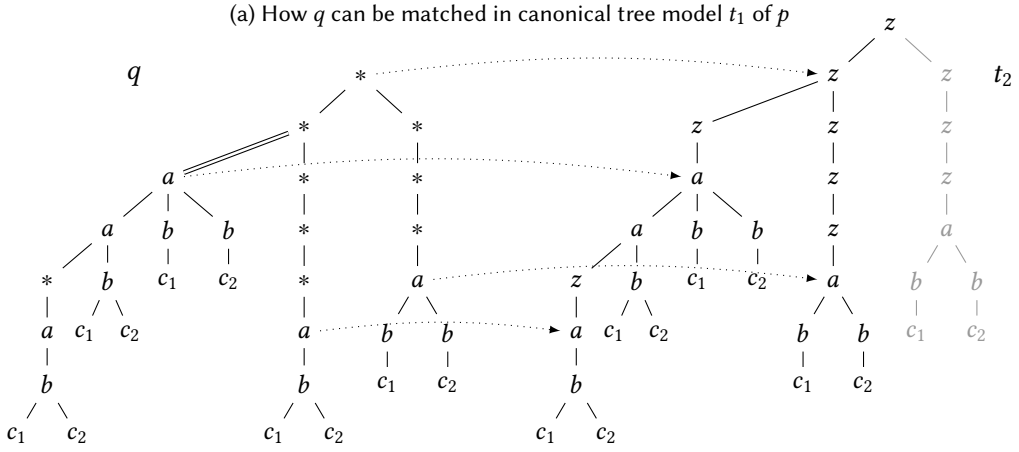
We define *relaxed patterns*, which generalize adorned tree patterns since they bring patterns in the same normal form if they have the same adorned tree patterns (such as those in Figure 12a), but also those in Figure 12b, which have different adorned tree patterns.

The *relaxed pattern* of a tree pattern p is obtained by replacing child edges by descendant edges in the following situations (until no replacements can be made anymore):

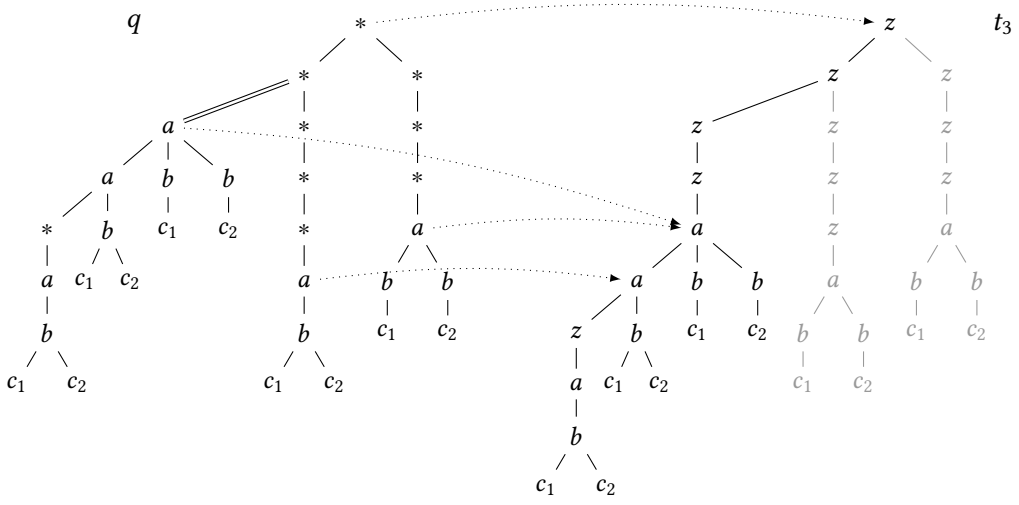
- when the child edge starts in a wildcard node that has no other outgoing edges, and that either has no parent or is connected to its parent via a descendant edge (as in Figure 12a), and
- when the child edge ends in a wildcard node such that no child edge starts in that node (as depicted in Figure 12b, where we give another example of two equivalent minimal tree patterns).



(a) How q can be matched in canonical tree model t_1 of p

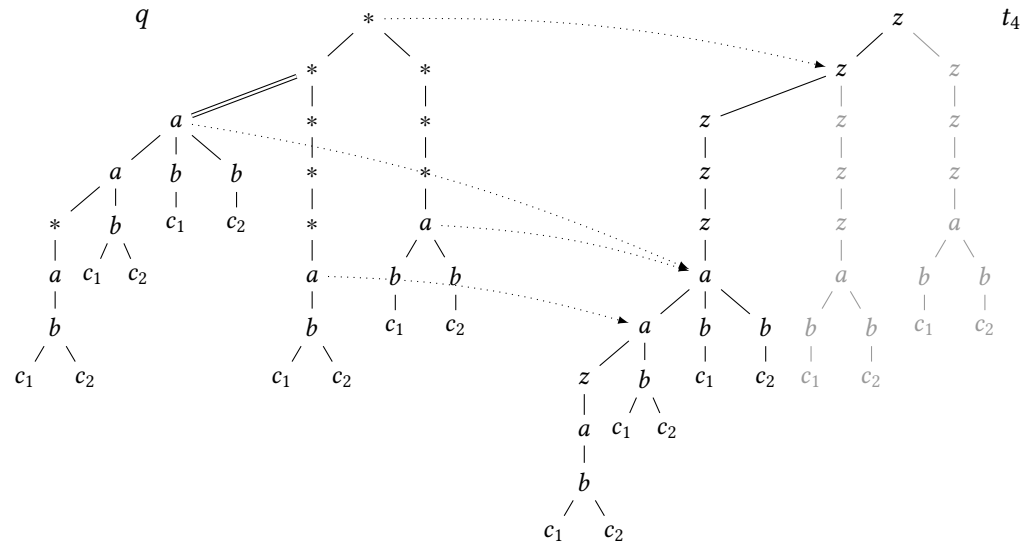


(b) How q can be matched in canonical tree model t_2 of p

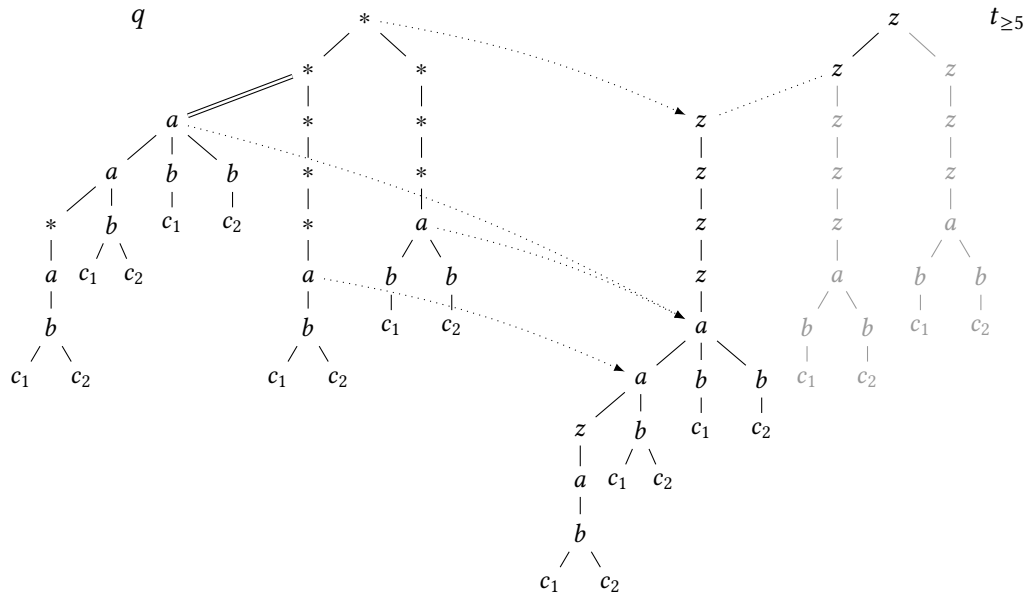


(c) How q can be matched in canonical tree model t_3 of p

Fig. 11 (part 1). Showing that patterns p and q in Figure 9 are equivalent



(d) How q can be matched in canonical tree model t_4 of p



(e) How q can be matched in all other canonical tree models of p

Fig. 11 (part 2). Showing that patterns p and q in Figure 9 are equivalent

It is easy to see that, given a tree pattern p, its relaxed pattern is unique. Indeed, changing a child edge to a descendant edge according to one situation will never invalidate one of the situations for another child edge.

The question in this section is whether there exist two equivalent minimal tree patterns that do not have isomorphic relaxed patterns. We can ask even more generally: are there two equivalent

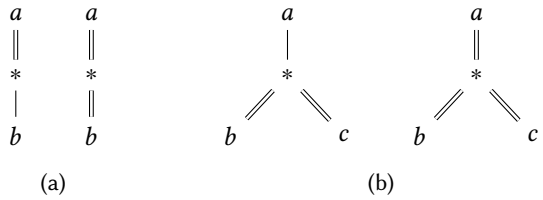


Fig. 12. Two examples of two equivalent minimal tree patterns

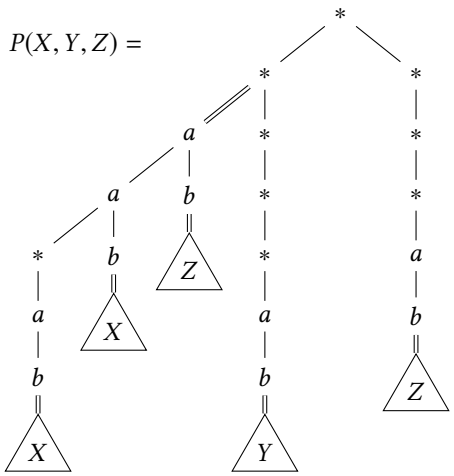


Fig. 13. Gadget for constructing tree patterns (Section 5.2 and Lemma 6.3)

minimal tree patterns that are *structurally different*, i.e., that differ even if we do not distinguish between child edges and descendant edges, and if we do not distinguish between different labels. We show that such patterns can be obtained as follows. Consider the gadget $P(X, Y, Z)$ in Figure 13. For tree patterns p, q, and r, denote by $P(p, q, r)$ the tree pattern obtained from P by instantiating the subpatterns marked X, Y, and Z by p, q, and r, respectively. An important detail of the gadget is that the subpatterns p, q and, r of $P(p, q, r)$ are connected to the b-nodes by descendant edges.

Consider the tree patterns p, q_1, q_2, and r from Figure 14. Then, we claim that

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

are equivalent and minimal, but structurally different. Indeed, q_1 and q_2 are structurally different, hence so are $P(p, q_1, r)$ and $P(p, q_2, r)$. 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 13.

Lemma 5.3. *Let p_X, p_{Y1}, p_{Y2}, and p_Z be tree patterns that do not use labels in $\{a, b\}$ and such that $p_X \subseteq p_{Y1} \subseteq p_Z$ and $p_X \subseteq p_{Y2} \subseteq p_Z$. Then $P(p_X, p_{Y1}, p_Z) \equiv P(p_X, p_{Y2}, p_Z)$.*

Proof. We only show that $P(p_X, p_{Y1}, p_Z) \subseteq P(p_X, p_{Y2}, p_Z)$ since the other direction is symmetric. The proof follows the same lines as the proof showing that the two tree patterns in Figure 9 are equivalent. By Lemma 4.1, it suffices to prove that $\text{Can}(P(p_X, p_{Y1}, p_Z)) \subseteq M_t(P(p_X, p_{Y2}, p_Z))$. To this end, take a canonical tree model t of $P(p_X, p_{Y1}, p_Z)$. We consider three cases, depending on

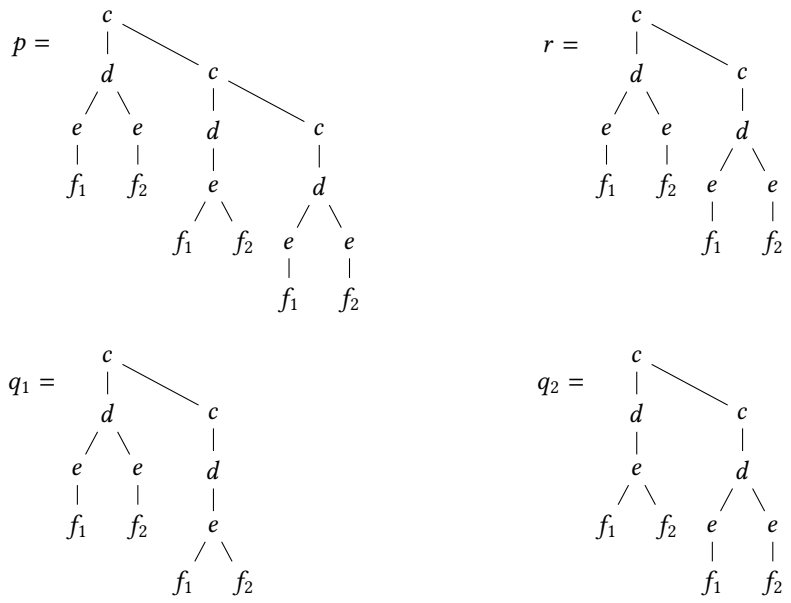


Fig. 14. Tree patterns used to show that minimal tree patters can be structurally different

whether the descendant edge of $P(p_X, p_{Y1}, p_Z)$ located above the a-node on depth 2 is replaced in t by one edge, two edges, or at least three edges. In each of these cases, we are going to match $P(p_X, p_{Y2}, p_Z)$ in t in a way analogous to that shown on Figure 11 for the previously considered patterns. As a part of such a match, we need a match of the subpatterns p_X, p_{Y2}, p_Z in appropriate places of the tree t. Namely, it is enough to

- match p_X in the canonical tree models of p_X being a part of t,
- match p_Z in the canonical tree models of p_Z being a part of t,
- match p_{Y2} in the canonical tree models of p_X being a part of t,
- match p_Z in the canonical tree model of p_{Y1} being a part of t.

This is possible by Lemma 4.1, because of the inclusions $p_X \subseteq p_X, p_Z \subseteq p_Z, p_X \subseteq p_{Y2}$, and $p_{Y1} \subseteq p_Z$. It was important here that the subpatterns p_X, p_{Y1}, p_{Y2}, p_Z are attached on descendant edges, not on child edges; thanks to that, it is enough to match p_Z anywhere in the canonical tree model of p_{Y1}, without the requirement that the root of p_Z has to be matched to the root of the canonical tree model (similarly for p_{Y2} and p_X). □

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

Proposition 5.4. *$P(p, q_1, r)$ and $P(p, q_2, r)$ are equivalent.*

Proof. The equivalence follows from Lemma 5.3 once we know that $p \subseteq q_1 \subseteq q$ and $p \subseteq q_2 \subseteq q$. These inclusions hold because there are homomorphisms from r to q_1, from r to q_2 (here we only glue together two e-nodes), from q_1 to p, and from q_2 to p (in the last case, the topmost c-node of q_2 is mapped to the middle c-node of p). □

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 6. More precisely, minimality of $P(p, q_1, r)$ and $P(p, q_2, r)$ follows from Lemma 6.2. In order to use this lemma, we have to know that p and r are minimal, and that

q_1 and q_2 are as small as possible, that is, there is no tree patterns q' with $p \subseteq q' \subseteq r$ and $\text{size}(q') < 11 = \text{size}(q_1) = \text{size}(q_2)$. Since p and r are $*$-narrow, their minimality is immediate because they are nonredundant (Lemma 2.9). It remains to prove the latter condition. To this end, suppose that there is some tree pattern q' such that $p \subseteq q' \subseteq r$ and $\text{size}(q') \leq 10$. Let $t_{q'}$ be the canonical tree model of q' in which every descendant edge of q' was replaced by two edges with a new z-node between them. Let m^r be an arbitrary match of r in $t_{q'}$, which exists because $q' \subseteq r$. All nodes of r are matched by m^r to nodes of $t_{q'}$ that existed already in q', because the newly created nodes have label z; thus $|\text{im}(\mathrm{m}^r)| \leq \text{size}(q')$, where $|\text{im}(\mathrm{m}^r)|$ denotes the number of nodes in the image of m^r. Moreover, we have that $\mathrm{m}^r(x) = \mathrm{m}^r(y)$ for $x \neq y$ only when x and y are on the same depth and have the same label, thus only when x and y are e-labeled siblings. The only way we can ensure $|\text{im}(\mathrm{m}^r)| \leq \text{size}(q') \leq 10$ is to match both nodes in both pairs of e-siblings to the same node, and to have in q' only those nodes that are in the image of m^r. Next, we notice that if (x, y) is a child edge in r, then $(\mathrm{m}^r(x), \mathrm{m}^r(y))$ is an edge in $t_{q'}$, and thus it is a child edge also in q'. This gives only one possible candidate for q': it is the pattern obtained from r by merging both pairs of e-siblings. We notice, however, that such a pattern q' does not satisfy $p \subseteq q'$, and thus actually no q' satisfying our assumptions can exist.

6 THE COMPLEXITY OF MINIMIZATION

In this section we present the main steps of the proof of the following lemma:

Lemma 6.1. *Tree Pattern Minimization is Σ_2^P-hard.*

In the proof, we will only consider tree models. More specifically, we prove that there exists a class of tree patterns such that, given a tree pattern p from this class and $k \in \mathbb{N}$, it is Σ_2^P-hard to decide if there exists a tree pattern q of size at most k for which $M_t(p) = M_t(q)$. By Proposition 2.4, we have that $M_t(p) = M_t(q)$ if and only if $M(p) = M(q)$, which proves the lemma.

Therefore, we only consider tree models from here on in the proof. The outline of the proof is as follows. We will first introduce an intermediate problem called *relative tree pattern minimization*. We will then perform 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. The first reduction will use a technical lemma for which we postpone the proof to Section 7.

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

We note that Relative Tree Pattern Minimization is a promise problem, i.e., the output is undefined if one of the patterns p and r is not minimal or if $p \subseteq r$ does not hold. Checking whether the promise (especially the minimality of p and r) is satisfied is as hard as solving the original problem. We need the conditions in the promise in the reduction from Relative Tree Pattern Minimization to Tree Pattern Minimization. However, the reduction showing that Relative Tree Pattern Minimization is hard always produces tree patterns p and r that we prove to be minimal and to satisfy $p \subseteq r$. Therefore, the composition of the two reductions is indeed a logarithmic space computable reduction.

We will use the gadget $P(X, Y, Z)$ from Figure 13. 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 subpatterns marked X, Y, and Z by p, q, and r, respectively.

The following lemma is crucial for proving that Relative Tree Pattern Minimization can be reduced to Tree Pattern Minimization.

Lemma 6.2. *Let p, q, and r be tree patterns that have non-wildcard roots, do not use labels in $\{a, b\}$, and such that $p \subseteq q \subseteq r$. If*

(1) p and r are minimal, and
(2) there is no tree pattern q' such that
- *$p \subseteq q' \subseteq r$ and*
- *$size(q') < size(q)$,*

then $P(p, q, r)$ is minimal.

We prove the lemma in Section 7. Notice that condition (2) in Lemma 6.2 is subtle. If $P(p, q, r)$ is minimal, then all p, q, r must also be minimal. But minimality of q does not 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 $\text{size}(q') < \text{size}(q)$ and $p \subseteq q' \subseteq r$, then $P(p, q', r)$ would be equivalent to $P(p, q, r)$ (by Lemma 5.3) and smaller.

Lemma 6.3. *For patterns p and r that have non-wildcard roots, 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 $p \subseteq r$, and a number $k \in \mathbb{N}$. We assume furthermore that p and r have non-wildcard roots. Since we can rename labels, we can also assume that p and r do not use the labels a and b. We will construct an instance p_m and $k' \in \mathbb{N}$ of Tree Pattern Minimization so that there exists a tree pattern of size at most k' equivalent to p_m if and only if there is a tree pattern q with $\text{size}(q) \leq k$ and $p \subseteq q \subseteq r$.

We define the tree pattern p_m to be $P(p, p, r)$, where the gadget P is illustrated in Figure 13. We define k' as $k + 2 \cdot \text{size}(p) + 2 \cdot \text{size}(r) + 19$.

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 5.3 that $p_m = P(p, p, r) \equiv P(p, q, r)$. Furthermore, the size of $P(p, q, r)$ is $\text{size}(q) + 2 \cdot \text{size}(p) + 2 \cdot \text{size}(r) + 19 \leq k'$.

We prove the other implication. Assume that $p_m = P(p, p, r)$ has an equivalent tree pattern of size at most k'. We want to prove that there exists a tree pattern q of size at most k such that $p \subseteq q \subseteq r$. Let q be a tree pattern such that $p \subseteq q \subseteq r$ and such that there exists no pattern q' smaller than q, for which $p \subseteq q' \subseteq r$ (of course such a pattern exists, since, e.g., $q := p$ satisfies $p \subseteq q \subseteq r$).

By Lemma 5.3, we have that $P(p, q, r) \equiv p_m$. The tree patterns p and r are minimal and q is a pattern of size $\min\{\text{size}(q') \mid p \subseteq q' \subseteq r\}$. Therefore, by Lemma 6.2, pattern $P(p, q, r)$ is minimal, and thus its size is at most $k' = k + 2 \cdot \text{size}(p) + 2 \cdot \text{size}(r) + 19$. This implies that $\text{size}(q) \leq k$. Clearly, the reduction can be performed in logarithmic space, which concludes the proof. □

We now show the second part of the reduction by proving that Relative Tree Pattern Minimization is Σ_2^P-complete. We will reduce from the following problem, which is a mild variation of $\exists\forall$-QBF, the canonical satisfiability problem of quantified Boolean $\exists\forall$-formulas.

$\exists$-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 $x_1, \ldots, x_n$?

We show that $\exists$-validity is Σ_2^P-complete by reduction from $\exists\forall$-QBF.

Lemma 6.4. *∃-validity is Σ_2^P-complete under logarithmic space reductions.*

Proof. Membership in Σ_2^P is obvious. For proving Σ_2^P-hardness, 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 an ∃∀-QBF formula such that $\Phi = c_1 \vee \cdots \vee c_m$ is quantifier-free and in disjunctive normal form. The problem of deciding whether such a formula is true is known to be Σ_2^P-hard [38, Theorem 4.1]. We compute the ∃-validity instance

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

This concludes the reduction, which can clearly be done in logarithmic space. For showing the correctness of the reduction, we first observe that Ψ is equivalent to

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

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 appears in the resulting formula (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 Lemma 6.1.

Lemma 6.5. *Relative Tree Pattern Minimization is Σ_2^P-complete under logarithmic space reductions, even if the patterns are restricted to have non-wildcard roots.*

Proof. The upper bound follows from the straightforward algorithm: guess q and check whether $p \subseteq q \subseteq r$. Clearly, guessing q can be done by a polynomial number of guesses. We never need to guess a pattern larger than size(r), since r trivially satisfies $p \subseteq r \subseteq r$. The containment tests can be done in coNP by Proposition 2.5.

For the lower bound, we reduce from ∃-validity. We borrow an idea from Miklau and Suciu's proof that containment of tree patterns is coNP-hard [29, Proofs of Lemma 3 and Theorem 4], but we have to make multiple significant changes. Let $I = \{(c_1^1, c_1^2), \ldots, (c_m^1, c_m^2)\}$ be an instance of ∃-validity. We can assume w.l.o.g. that no clause contains the same variable twice. Indeed, such clauses are either unnecessarily large ($x_i \wedge x_i$) or not satisfiable ($x_i \wedge \neg x_i$). We compute the patterns p and r as shown in Figure 15, and we let $k = \text{size}(r) - m$. Obviously, both patterns have non-wildcard roots. Each subpattern of r that is rooted in 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 is connected to a sequence of three nodes connected by descendant edges; the middle node is labeled by $*$, and the two other nodes by x_i (as in Figure 15).

Notice that the pattern p only depends on the *number* of clauses and variables of I and not on the clauses themselves (it uses m in the picture of p and n in the subpatterns C and D). Furthermore, p does not contain any wildcards and contains descendant edges only in its subpattern D. In consequence the canonical tree models of p only differ from p in the subtree corresponding to 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.

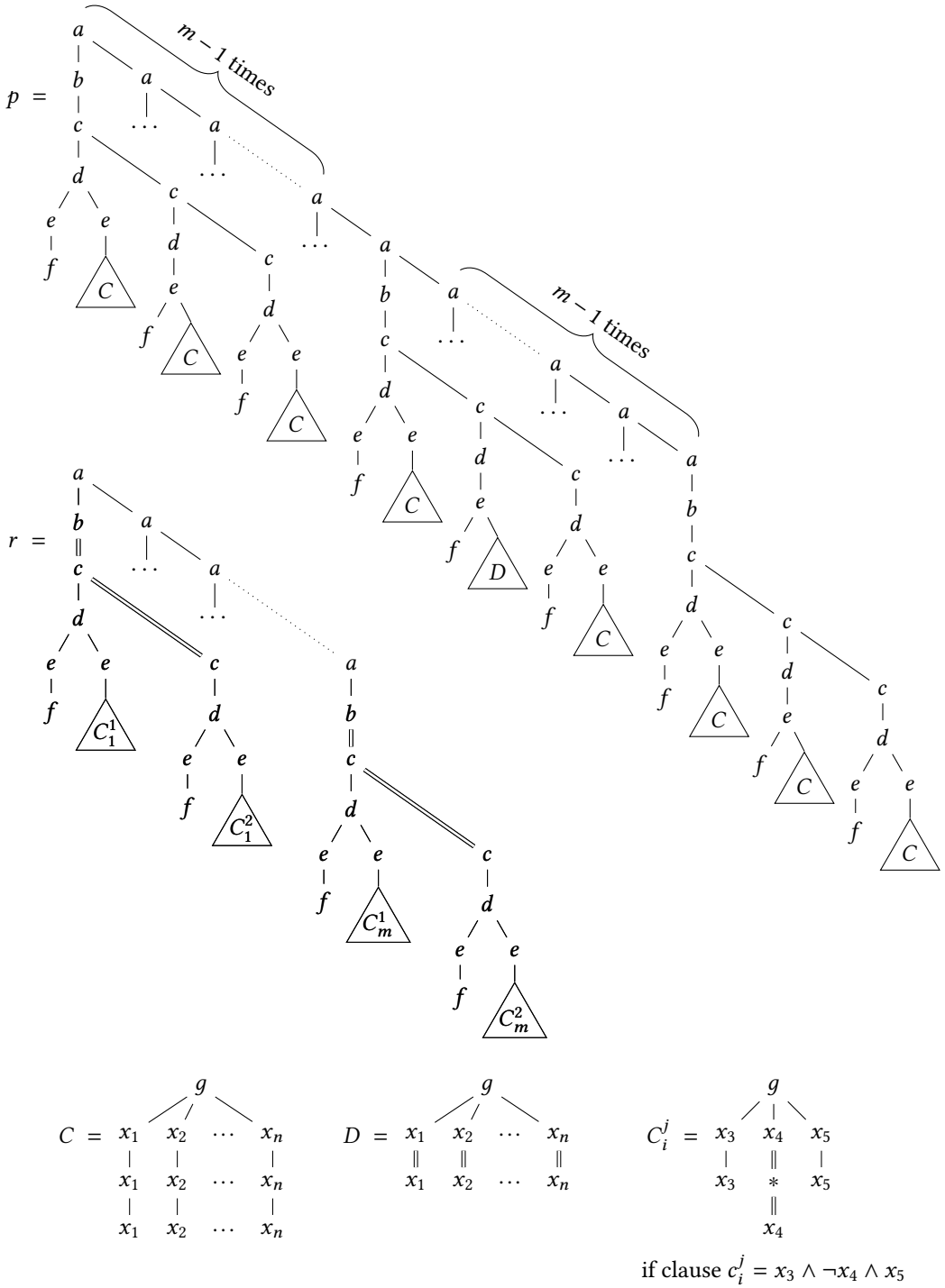


Fig. 15. Tree patterns used in the proof of Lemma 6.5

Clearly, the patterns p and q and the number k can be computed using logarithmic space. We will show that I is a yes-instance of $\exists$-validity if and only if p, r, and k are a yes-instance of Relative Tree Pattern Minimization. Before that, we prove that p, r, and k are at all a correct instance of Relative Tree Pattern Minimization, which is a consequence of the following claim.

Claim 6.6. *The tree patterns p and r are minimal and $p \subseteq r$.*

We prove the claim. We first show that p and r are minimal. Since p and r both are $*$-narrow, it suffices to show that both patterns are nonredundant, according to Lemma 2.9. It is easy (but tedious) to verify that removing any leaf from any canonical tree of p results in a tree that is not in $M_t(p)$. By Lemma 4.1, 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 that r is minimal is analogous.

We now prove that $p \subseteq r$. We show that there is a homomorphism from r to p, which proves that $p \subseteq r$ (Observation 4.2). It is easy to see that such a homomorphism exists: the root of r can be mapped to the root of p and each C_i^j-subpattern in r can be mapped to a C-subpattern in p. This concludes the proof of Claim 6.6. □

Next, we show that if I is a yes-instance of $\exists$-validity, then p, r, and k are a yes-instance of Relative Tree Pattern Minimization. To this end, suppose that I is a yes-instance of $\exists$-validity, and fix a choice of clauses $\iota\colon \{1,\dots,m\} \to \{1,2\}$ such that $c_1^{\iota(1)} \vee \dots \vee c_m^{\iota(m)}$ is true for all valuations of variables. We obtain q from r by the following algorithm: for every $i \in \{1,\dots,m\}$ take the e-labeled parent of the subpattern $C_i^{\iota(i)}$ and merge it with its e-labeled sibling.

Because we have merged exactly m nodes, we have $\mathrm{size}(q) = \mathrm{size}(r) - m = k$. We also see that $q \subseteq r$, since the operation of merging sibling nodes only imposes extra restrictions on the set of models.

It remains to show that $p \subseteq q$ holds, which we do by showing that there is a match $\mathrm{m}^{q,t}$ of q in every canonical tree model t of p. To this end, let t be an arbitrary canonical tree model of p and let a_m be the middle a-labeled node in t. With t we associate a valuation σ^t of the variables $x_1,\dots,x_n$ as follows:

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

where $d^t(x_i)$ denotes the distance between the two x_i-labeled nodes in the subtree of t corresponding to the D-subpattern of p.

Since $c_1^{\iota(1)} \vee \dots \vee c_m^{\iota(m)}$ is universally true, there must be a k such that the clause $c_k^{\iota(k)}$ is satisfied by σ^t. We define $\mathrm{m}^{q,t}$ such that

- the k-th a-labeled node of q (i.e., on depth $k-1$) is mapped to a_m and[10]
- for every subpattern of q rooted at a b-labeled node, the d-labeled nodes are matched so that the d-labeled node with only one child is mapped to the d-labeled node with only one child in the corresponding subtree of t.

From this point on, matching of the a-, b-, c-, d-, e-, f-, and g-labeled nodes is obvious, as there is only one way to do it. Furthermore, each of the C_i^j-subpatterns can be matched in one of the copies of C. It remains to show that $C_k^{\iota(k)}$ can be matched in the part t_D of t corresponding to D.

[10] In other words: the first a-labeled node of q is mapped to the $(m-k+1)$-th a-labeled node of t, counting from the root downwards.

If, in $C_k^{\iota(k)}$, below g there exist two x_i-nodes being in distance one, then x_i appears positively in $c_k^{\iota(k)}$. Therefore, $\sigma^t(x_i)$ has to be true (as $c_k^{\iota(k)}$ is satisfied) and we can conclude that $d^t(x_i) = 1$ by definition of σ^t. As $d^t(x_i) = 1$, we can match the two x_i-labeled nodes in t_D.

On the other hand, if, in $C_k^{\iota(k)}$, below g there exist two x_i-nodes with a wildcard node between them, then $\neg x_i$ appears in $c_k^{\iota(k)}$. Therefore, $\sigma^t(x_i)$ has to be false and we can conclude that $d^t(x_i) > 1$ by definition of σ^t. As $d^t(x_i) > 1$, we again can match the two x_i-labeled nodes in t_D (including the wildcard node between these nodes). From the construction, it is clear that $\mathrm{m}^{q,t}$ is indeed a match of q in t. This shows that p, r, and k are indeed a yes-instance to Relative Tree Pattern Minimization.

We continue by proving the more difficult direction: if p, r, and k are a yes-instance of Relative Tree Pattern Minimization, then I is a yes-instance of $\exists$-validity. We first need to limit the form of candidate solutions q to Relative Tree Pattern Minimization. We say that a tree pattern q is *well-formed* if it can be obtained from pattern r by the following algorithm: Select in each subpattern rooted at a b-labeled node one subpattern rooted at a d-labeled node. In each selected subpattern, merge the two nodes labeled e—other nodes remain unchanged. Furthermore, we allow to replace descendant edges by child edges and to replace wildcards by other labels.[11]

Claim 6.7. *Let q be a pattern such that $p \subseteq q \subseteq r$ and $\mathrm{size}(q) \leq \mathrm{size}(r) - m$. Then q is well-formed.*

We prove the claim. We use the following fact that holds for all matches m of r (into arbitrary trees t) and all nodes $u \neq v$ of r:

$$\mathrm{m}(u) = \mathrm{m}(v) \text{ implies that } \mathrm{lab}(u) = \mathrm{lab}(v) = e \text{ and that } u \text{ and } v \text{ are siblings.} \qquad (\star)$$

The fact ($\star$) can be verified by observing that all other pairwise different nodes have different sequences of labels on the path to the root of r. Therefore these nodes must also be matched on different paths in t.

We continue with the proof of the claim. Let t_q be the canonical tree model of q obtained as follows: every descendant edge connecting two x_i-nodes (for some i) is replaced by a single edge; every other descendant edge is replaced by two edges with a new z-labeled node between them. Let m^r be an arbitrary match of r in t_q. All non-wildcard nodes of r are mapped to nodes of t_q that existed already in q. Consider a wildcard node of r. It is located between two x_i-nodes. It is possible that this wildcard node is mapped to a z-node of t_q, created in the middle of some descendant edge of q. But at least one end of this edge is not labeled by x_i (we have not added z-nodes on descendant edges connecting two x_i-nodes), and thus is not used as the image of the parent (resp., the child) of the considered wildcard node. In consequence, we can alter m^r so that it maps the wildcard node to this end of the descendant edge. This means that we can assume w.l.o.g. that the image of m^r contains only nodes of q. We can thus consider m^r as a mapping from r to q. We also notice that if (u, v) is a child edge in r, then there is an edge $(\mathrm{m}^r(u), \mathrm{m}^r(v))$ in t_q, and thus also in q. If u is not labeled by any x_i, then this edge in q is a child edge. If (u, v) is a descendant edge in r, we only know that $\mathrm{m}^r(v)$ is a descendant of $\mathrm{m}^r(u)$ in q.

Let $|\mathrm{im}(\mathrm{m}^r)|$ denote the number of nodes in the image of m^r, i.e., the cardinality of $\{u \in t \mid \exists v \in r.\ \mathrm{m}^r(v) = u\}$. By the assumptions of the claim, we have $|\mathrm{im}(\mathrm{m}^r)| \leq \mathrm{size}(q) \leq \mathrm{size}(r) - m$. This implies, due to ($\star$), that there are at least m pairs of e-siblings that are matched by m^r to the same node. We can observe that in every subpattern rooted at a b-labeled node in r, at most one pair of e-siblings can be matched by m^r to the same node. Indeed, suppose to the contrary that both pairs

[11] 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 and replacement of wildcards by other labels is because we cannot avoid it in Claim 6.7.

of e-siblings below some b-node in r are matched by m^r to the same node. Consider these e-nodes, their f- and g-children, their d-parents, and their c-grandparents. The image of these nodes in q consists of an e-node with an f-child, a g-child, a d-parent, and a c-grandparent (connected by child edges only), and of another copy of such a tree fragment, where the c-grandparent in one of the copies is a descendant of the c-grandparent in the other copy. Such a fragment of q cannot be matched in the smallest canonical tree model t_p of p. This is because t_p has, in each subtree rooted at a b-labeled node, at most one e-node having both an f-child and a g-child (and it is impossible to map both considered e-nodes of q to the same node of t_p). So this would contradict that $p \subseteq q$. In consequence, below every b-labeled node exactly one pair of e-siblings is matched by m^r to the same node, and $|\mathrm{im}(\mathrm{m}^r)| = \mathrm{size}(r) - m = \mathrm{size}(q)$.

This means that there are no other nodes in q than those coming from r. Consider now a descendant edge (u, v) of r. We need to argue that there is an edge $(\mathrm{m}^r(u), \mathrm{m}^r(v))$ in q, as right now we only know that $\mathrm{m}^r(v)$ is a descendant of $\mathrm{m}^r(u)$, and that, by surjectivity of m^r, all nodes between $\mathrm{m}^r(u)$ and $\mathrm{m}^r(v)$ are in the image of m^r. We have three series of descendant edges in r: those starting in a b-node, those starting in a c-node, and those contained in the C_i^j subpatterns. When u is a b-node, it is easy to see that the only node of r that can be matched to a child of $\mathrm{m}^r(u)$ is v (i.e., the c-child of u). Suppose that u is a c-node. This time, the only nodes of r that can be matched to a child of $\mathrm{m}^r(u)$ are v and the d-child w of u. If v (being a c-node) was matched to a descendant of $\mathrm{m}^r(w)$, then such q could not be matched in the smallest canonical tree model t_p of p, because in p there is no c-node located below a d-node. Thus v is matched to a child of $\mathrm{m}^r(u)$. Finally, if u is an x_i-node or a wildcard node, it is again easy to see (taking into account the previously resolved case) that the only node of r that can be matched to a child of $\mathrm{m}^r(u)$ is v.

Having the above, we almost know that q is well-formed, but we still need to show that if (u, v) is a child edge of r with u labeled by some x_i, then $(\mathrm{m}^r(u), \mathrm{m}^r(v))$ is a child edge in q. But, indeed, if $(\mathrm{m}^r(u), \mathrm{m}^r(v))$ was a descendant edge in q, then q would have a canonical tree model in which this edge is replaced by a sequence of edges; such a model could not be matched by r, contrary to the assumption $q \subseteq r$. This finishes the proof of Claim 6.7. □

We now proceed with the proof of Lemma 6.5. To this end, consider a pattern q such that $p \subseteq q \subseteq r$ and $\mathrm{size}(q) \leq k$; recall that $k = \mathrm{size}(r) - m$. By Claim 6.7, q is well-formed.

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 $\mathrm{f}^q \colon \{1, \ldots, m\} \to \{1, 2\}$ 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 as q is well-formed.

We are also going to use the valuation σ^t and the numbers $d^t(x_i)$ defined during the proof of the opposite direction for a canonical tree t of p.

We claim that

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

is valid (i.e., true for all valuations of variables). More precisely, for every canonical tree model t of p, we claim that the valuation σ^t satisfies $c_1^{\mathrm{f}^q(1)} \vee \cdots \vee c_m^{\mathrm{f}^q(m)}$. This shows validity of the formula because, for each valuation ρ of $x_1, \ldots, x_n$, there exists a canonical tree model t of p such that $\sigma^t = \rho$.

Let thus t be a canonical tree model of p and let m^q be a match of q in t. Notice that m^q exists because $p \subseteq q$. We denote the subtree of t corresponding to the subpattern D of p by t_D. Since there is a path of m a-labeled nodes in q (because q is in well-formed) and a path of $2m - 1$ a-labeled

nodes in t, exactly one a-labeled node of q has to be matched to the middle a-labeled node of t. Call this node a_j (and assume that it has depth $j-1$ in q, where the root has depth 0).

We show that the clause $c_j^{\mathrm{f}^q(j)}$ is satisfied by σ^t. To achieve this, it is sufficient to look how the subpattern rooted at the b-child of a_j is matched in the subtree rooted at the b-child of the middle a-labeled node of t (it has to be matched there, because the b-child of a_j is connected to a_j by a child edge). Consider this subtree of t and note that it only has a single e-labeled node that has both an f-labeled and a g-labeled child. Therefore, we know that m^q needs to match $C_j^{\mathrm{f}^q(j)}$ to t_D. We show that every literal of $c_j^{\mathrm{f}^q(j)}$ is satisfied by σ^t.

Let x_i be a positive literal of $c_j^{\mathrm{f}^q(j)}$. Then there are two x_i-labeled nodes in $C_j^{\mathrm{f}^q(j)}$, which are connected by a child edge. These nodes have to be matched to the x_i-labeled nodes of t_D. Therefore $d^t(x_i) = 1$ and $\sigma^t(x_i) = \text{true}$.

Now let $\neg x_i$ be a negative literal of $c_j^{\mathrm{f}^q(j)}$. Then there are two x_i-labeled nodes in $C_j^{\mathrm{f}^q(j)}$, connected by a path of two descendant edges. Again, these nodes have to be matched to the x_i-labeled nodes of t_D. Therefore $d^t(x_i) > 1$ and $\sigma^t(x_i) = \text{false}$. This concludes the proof that I is a yes-instance of ∃-VALIDITY, and thus the proof of Lemma 6.5. □

Finally we note that, throughout the entire proof of Lemma 6.1, we never use edge labels. In particular, the entire proof goes through just the same if every edge on every query carries the same, fixed label. We therefore have the following corollary.

COROLLARY 6.8. *GENERALIZED TREE PATTERN MINIMIZATION is Σ_2^P-hard, even if the patterns have no wildcard edges.*

Minimality. The following problem is closely related to TREE PATTERN MINIMIZATION and has also been considered in the literature:

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

Indeed, MINIMALITY can be seen as a variant of TREE PATTERN MINIMIZATION where $k = \text{size}(p) - 1$.

MINIMALITY is known to be NP-hard [26, Theorem 6.3] but, just as TREE PATTERN MINIMIZATION, its precise complexity was unknown. The techniques we used for proving that TREE PATTERN MINIMIZATION is Σ_2^P-complete can be used to prove that MINIMALITY is Π_2^P-complete.

THEOREM 6.9. *MINIMALITY is Π_2^P-complete.*

PROOF. 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 can use essentially the same (combined) reduction as in the proofs of Lemmas 6.5 and 6.3. Let I be an instance of ∃-VALIDITY (having m pairs of clauses) and let p and r be the patterns computed in the reduction from ∃-VALIDITY to RELATIVE TREE PATTERN MINIMIZATION in the proof of Lemma 6.5.

We compute a pattern q_0 from r by merging the e-labeled nodes above the subpatterns C_1^1 to C_{m-1}^1 with their siblings.

We show that $p \subseteq q_0 \subseteq r$. The inclusion $p \subseteq q_0$ holds because there exists a homomorphism from q_0 to p which maps the lowest subpattern rooted at a b-labeled node of q_0 to the subpattern in p rooted at the b-labeled ancestor of the D-subpattern. (The two c-labeled nodes from the q_0

pattern can be matched to the upper and lower c-labeled nodes inside the subpattern with D in p.) The inclusion $q_0 \subseteq r$ is immediate because q_0 is obtained by merging nodes of r, which imposes extra restrictions.

From the proofs of Lemmas 6.5 and 6.3 we know that I is a yes-instance if and only if there exists a tree pattern equivalent to $P(p, p, r)$ and having size at most $k = \text{size}(P(p, p, r)) - m$. Notice that $\text{size}(q_0) = \text{size}(p) - m + 1$, hence $k = \text{size}(P(p, q_0, r)) - 1$. By Lemma 5.3, we know that $P(p, p, r) \equiv P(p, q_0, r)$, and thus I is a yes instance if and only if there exists a tree pattern equivalent to $P(p, q_0, r)$ and having size at most $\text{size}(P(p, q_0, r)) - 1$. The latter condition simply says that $P(p, q_0, r)$ is not minimal. Thus, in order to solve I we just need to ask whether the pattern $P(p, q, r)$ is minimal, and negate the answer. □

7 TECHNICAL CORE

This section is dedicated to prove Lemma 6.2. We present a few general lemmas in Section 7.1 and lemmas specific to the structure of the tree pattern in Figure 13 in Section 7.2. We prove Lemma 6.2 in Section 7.3.

7.1 General Tools

We first revisit the definitions of match, containment, equivalence, and nonredundancy for tree models. The reason is that we will sometimes require in proofs that the root of the pattern is mapped to the root of the tree. We therefore say that a match m of tree pattern $p = (V_p, E_p, \text{lab}_p)$ into a tree $t = (V, E, \text{lab})$ is a *strong match* if $\text{m}(r_p) = r$, where r_p and r are roots of p and t, respectively. We define *strong containment*, *strong equivalence* and *strong redundancy* analogously to containment, equivalence and redundancy, but we require the matches in the respective definitions to be strong. For two tree patterns p and q we denote by $p \subseteq_S q$ that p is strongly contained is q and by $p \equiv_S q$ that p and q are strongly equivalent.

We note that strong redundancy implies redundancy, and thus for nonredundancy the implication is inverted: nonredundancy implies non-strong-redundancy. This implication can be strengthened a bit: if a pattern p is nonredundant, then every subpattern p^v is non-strongly-redundant. Indeed, towards a contradiction, if $p^v \equiv_S p^v \setminus n$ for some node n of p^v, then in every match of $p \setminus n$ in some tree t we can replace the match of the subpattern $p^v \setminus n$ by a match of p^v that matches v to the same node, obtaining a match p in t. As obviously $p \subseteq p \setminus n$, this shows that $p \equiv p \setminus n$, contradicting the assumption that p is nonredundant.

We have an analogue of Lemma 4.1 that holds for strong containment.

Lemma 7.1 ([29, Proposition 3]). *Let p and q be tree patterns. Then $p \subseteq_S q$ if and only if every canonical tree model of p is strongly matched by q.*

Every strong match is a match, and thus it immediately follows from Lemmas 4.1 and 7.1 that, if $p \subseteq_S q$ then also $p \subseteq q$ (and if $p \equiv_S q$ then also $p \equiv q$). The converse is not true in general. For example, the pattern $* \Rightarrow a$ is equivalent to $* \longrightarrow a$ but not strongly equivalent. The following result shows, however, that we can deduce strong equivalence from equivalence in a special case.

Lemma 7.2 (Corollary of [26, Theorem 4.3, case 1]). *Let p be a tree pattern such that the root of p is not a wildcard node. Then, for every tree pattern q we have that $p \equiv q$ if and only if $p \equiv_S q$.*

We now prove a few lemmas that are useful to infer similarities between equivalent patterns. We start by comparing roots.

Lemma 7.3. *Let p and q be tree patterns such that $p \equiv q$. Then the roots of p and of q have the same label.*

Proof. If the root of any of these patterns is not a wildcard node, by Lemma 7.2 we have $p \equiv_S q$, which implies that p has to strongly match in every canonical tree of q. This is only possible when the roots of p and of q have the same label. The remaining case is that both roots are wildcard nodes, for which the conclusion also holds. □

The next result, coming from [26], allows us to descend to the root's children. Recall that for a node v of p, by p^v we denote the subpattern rooted at v, and when v is a child of the root of p, by p_v we denote the subpattern consisting of p^v and of the root.

Lemma 7.4. *Let p and q be nonredundant and let $p \equiv q$. Then the roots of p and of q have the same number of children. Moreover, there exists a bijection φ between children of p's root and children of q's root such that for every child u of p's root it holds that $p_u \equiv q_{\varphi(u)}$.*

sketch. Lemma 4.11 in [26] is phrased differently from the present lemma and, in particular, its second condition is about *relative containment*. However, as Kimelfeld and Sagiv explain after Definition 3.11, in this case, containment in both directions (and therefore equivalence) is implied by the two directions of relative containment. □

We can prove an analogous result for strong equivalence and non-strong-redundancy (here some assumptions are stronger, but the thesis is also stronger).

Lemma 7.5. *Let p and q be non-strongly-redundant and let $p \equiv_S q$. Then the roots of p and of q have the same number of children. Moreover, there exists a bijection φ between children of p's root and children of q's root such that for every child u of p's root it holds that $p_u \equiv_S q_{\varphi(u)}$.*

Before proving Lemma 7.5, we recall the following result that will be useful in the proof.

Lemma 7.6 ([20, Lemma 4.4]). *Let p and q be tree patterns. Then $p \subseteq_S q$ if and only if, for each child v of q's root there exists a child u of p's root such that $p_u \subseteq_S q_v$.*[12]

of Lemma 7.5. Assume that p's root has n children denoted $u_1, \ldots, u_n$ and q's root has m children denoted $v_1, \ldots, v_m$. Since $p \subseteq_S q$, by Lemma 7.6, for every q_v (where $v \in \{v_1, \ldots, v_m\}$), there exists a p_u (where $u \in \{u_1, \ldots, u_n\}$) such that $p_u \subseteq_S q_v$. Let f be a function that maps each such v to one such corresponding u. Similarly we can define a function g such that for every $u \in \{u_1, \ldots, u_n\}$, we have that $g(u) \in \{v_1, \ldots, v_m\}$ satisfies $q_{g(u)} \subseteq_S p_u$.

We will show that $f \circ g$ is the identity. Assume otherwise; say that $f(g(u)) = u'$ for some $u' \neq u$. Then

$$p_{u'} = p_{f(g(u))} \subseteq_S q_{g(u)} \subseteq_S p_u \ .$$

However, this means that p is strongly redundant. Indeed: by removing the subpattern p^u we would obtain a strongly equivalent pattern. This is a contradiction. Similarly we obtain that $g \circ f$ is the identity.

We thus have $f = g^{-1}$, which means in particular that the roots of p and of q have the same number of children and that there is a bijection φ (defined as $\varphi(u) = g(u)$) such that

$$\forall u \in \{u_1, \ldots, u_n\} \colon p_u \subseteq_S q_{\varphi(u)} \text{ and } p_u \supseteq_S q_{\varphi(u)}$$

which simply means that

$$\forall u \in \{u_1, \ldots, u_n\} \colon p_u \equiv_S q_{\varphi(u)}$$

This concludes the proof. □

[12] When comparing this lemma to the statement of Flesca et al., it is important to note that Flesca et al. define matches such that the root of the pattern always needs to be matched to the root of the tree [20, page 7, definition of *embedding*]. Therefore, their notion of containment corresponds to our notion of strong containment.

The next two lemmas allow to preserve equivalence while removing the root (the first of them works already for containment).

Lemma 7.7. *Let p and q be two tree patterns such that $p \subseteq q$ and the roots of p and q have exactly one child (u and v, respectively). Then $p^u \subseteq q^v$.*

Proof. Let $t' \in \mathrm{Can}(p^u)$. Let t be a tree obtained from t' by adding a new root above the root of t' and labeling it by the label of the root of p, or by z if the root of p is labeled $*$. Clearly we have that $t \in \mathrm{Can}(p)$, so p can be matched in t. By $p \subseteq q$, there exists a match of q in t. In this match, the image of v is a node below the root of t, that is, a node inside t'. This means that q^v matches in t', which shows that $p^u \subseteq q^v$. □

Under a slightly stronger assumption than in Lemma 7.7, we can even obtain strong equivalence of the subpatterns, as stated in the next lemma.

Lemma 7.8. *Let p and q be two tree patterns such that $p \equiv_S q$ and the roots of p and q have exactly one child (u and v, respectively). Additionally, suppose that*

(a) u is not a wildcard node, or
(b) the edges from p's root to u and from q's root to v are child edges.

Then $p^u \equiv_S q^v$ and the edges from p's root to u and from q's root to v are either both child edges, or both descendant edges.[13]

Proof. In case (a), we first deduce $p^u \equiv q^v$ from Lemma 7.7. This can be strengthened to $p^u \equiv_S q^v$ by Lemma 7.2, due to the assumption that the root of p^u is not a wildcard node. Suppose now, to the contrary of our conclusion, that the edge from p's root to u is a child edge but the edge from q's root to v is a descendant edge. Then we can consider a canonical tree model t of q in which this descendant edge is replaced by a path of length at least 2. The (only) root's child in t has label z, and thus p cannot be strongly matched in t, which contradicts our assumption that $p \equiv_S q$. In the situation where the edge in p is a descendant edge and the edge in q is a child edge, we know by Lemma 7.3 that v is not a wildcard node. Therefore, this case is resolved by symmetry. Thus both edges are of the same type.

Consider now case (b). Let $t \in \mathrm{Can}(q^v)$ be an arbitrary canonical tree model for q^v. By definition of q, there is a tree t', strongly matched by q, where the only difference to t is that t' has a new root with the root of t being its child. As $p \equiv_S q$, p strongly matches in t' and as the edge from p's root to u is a child edge, p^u strongly matches in t. By Lemma 7.1 we get that $q^v \subseteq_S p^u$. The proof for $p^u \subseteq_S q^v$ is completely symmetric, yielding $p^u \equiv_S q^v$ as desired. □

The next lemma is similar to Lemma 7.6, but talks about nodes on an arbitrary depth.

Lemma 7.9. *Let $k \in \mathbb{N}$, and let p and q be two tree patterns such that $p \subseteq q$. Then, for every node v on depth k in q, there exists a node u on depth k in p such that $p^u \subseteq q^v$.*

Proof. Assume towards a contradiction that there exists a node v on depth k in q such that there is no node u on depth k in p with $p^u \subseteq q^v$. Let $t \in \mathrm{Can}(p)$ be a canonical tree model of p such that

(a) every descendant edge on depth smaller than k is replaced with a single child edge; and
(b) for every node w on depth k in t, it holds that q^v does not match t^w.

By assumption and by Lemma 4.1, such a canonical tree model exists. By (b), we know that q^v also cannot be strongly matched in any subtree of t starting on depth at least k. Therefore, q cannot be matched in t, which contradicts the assumption $p \subseteq q$. □

[13] Of course the latter part of the conclusion is nontrivial only in case (a).

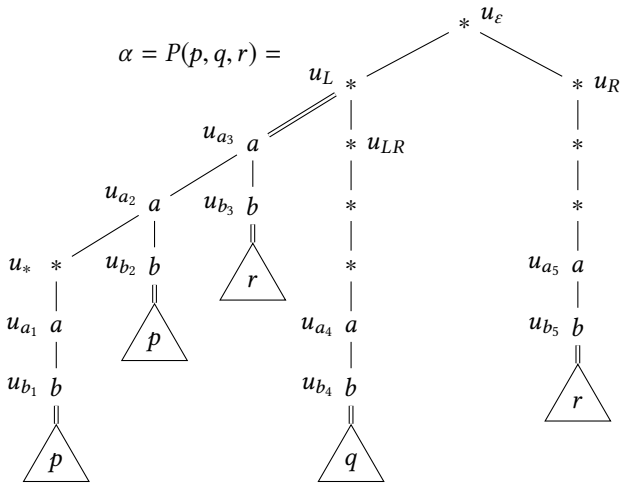


Fig. 16. The main gadget, instantiated with patterns p, q, and r, and annotated with node names for the proof of Lemma 6.3

7.2 Gadget-Specific Tools

Before we prove Lemma 6.2, we do a first step by showing that the pattern $P(p, q, r)$ is nonredundant if the conditions (1) and (2) of Lemma 6.2 are satisfied.

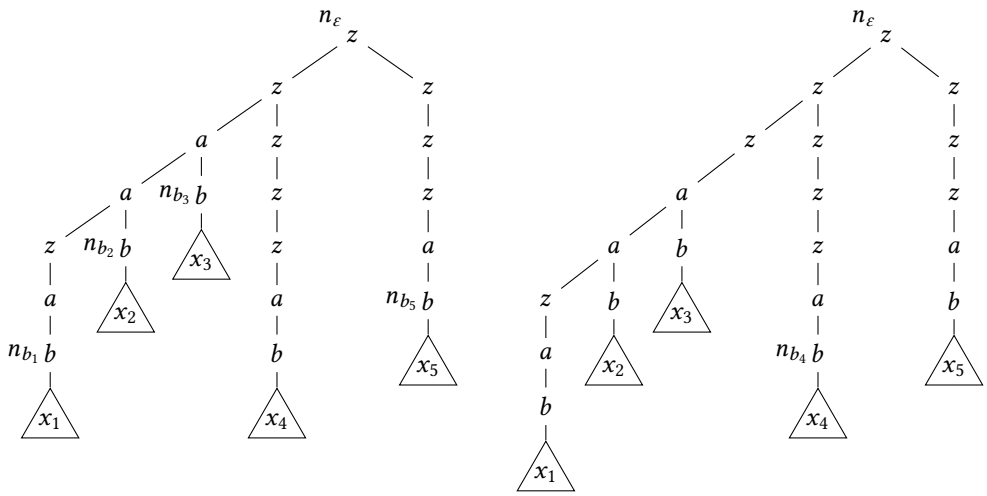
We repeated the gadget from Figure 13 in Figure 16, where we annotate it with node names to facilitate references.

Definition 7.10. We say that a pattern p is *barely contained* in a pattern q if $p \subseteq q$ but, for every leaf n of p, it holds that $p \setminus n \not\subseteq q$. We denote it by $p \subseteq_B q$.

Lemma 7.11. *Let p, q, and r be nonredundant tree patterns not using labels a nor b, and such that $q \subseteq_B r$. Then $P(p, q, r)$ is nonredundant.*

Proof. The proof is analogous to the proof from page 16 showing that the pattern p in Figure 9 is nonredundant. Let $\alpha = P(p, q, r)$. Throughout the proof, we use $u_{b_1}, \ldots, u_{b_5}$ to denote the b-nodes of α from left to right. We show that, for every leaf u of α, we have $\alpha \setminus u \not\equiv \alpha$. We assume towards a contradiction that $\alpha \setminus u \equiv \alpha$ for some leaf u of α. The proof is by a case distinction depending on the i for which u is a descendant of u_{b_i}.

For the proof we will use something similar as our main gadget, but this time with five parameters. More precisely, the two structures $T_j(x_1, \ldots, x_5)$ for $j = 1, 2$ in Figure 17 can be instantiated with 5-tuples of trees $(t_1, \ldots, t_5)$. The result is a tree $T_j(t_1, t_2, \ldots, t_5)$ obtained from T_j by replacing each x_k by t_k. These trees are canonical tree models of α if the subtrees $t_1, \ldots, t_5$ are canonical tree models of p, p, r, q, and r, respectively. Let t_x, for every $x \in \{p, q, r\}$, be a canonical tree model of x and $t_{x \setminus u}$, for each leaf u of x, be a canonical tree model of $x \setminus u$ such that $t_{x \setminus u} \notin M_t(x)$. Furthermore, if $x = q$, we require that $t_{x \setminus u} \notin M_t(r)$. As p, q, and r are nonredundant and $q \subseteq_B r$ (which implies



(a) $T = T_1(x_1, x_2, x_3, x_4, x_5)$ (b) $T = T_2(x_1, x_2, x_3, x_4, x_5)$

Fig. 17. Gadget for making canonical tree models for the proof of Lemma 7.11

$q \setminus u \not\subseteq r$ and $q \subseteq r$), such trees exist. We choose a tree $t \in M_t(\alpha \setminus u)$ as follows:

$$t = \begin{cases} T_1(t_{p\setminus u},\ t_p\ ,\ t_r\ ,\ t_q\ ,\ t_r\ ) & \text{if } u \text{ is below } u_{b_1} \\ T_1(t_p\ ,\ t_{p\setminus u},\ t_r\ ,\ t_q\ ,\ t_r\ ) & \text{if } u \text{ is below } u_{b_2} \\ T_1(t_p\ ,\ t_p\ ,\ t_{r\setminus u},\ t_q\ ,\ t_r\ ) & \text{if } u \text{ is below } u_{b_3} \\ T_2(t_p\ ,\ t_p\ ,\ t_r\ ,\ t_{q\setminus u},\ t_r\ ) & \text{if } u \text{ is below } u_{b_4} \\ T_1(t_p\ ,\ t_p\ ,\ t_r\ ,\ t_q\ ,\ t_{r\setminus u}) & \text{if } u \text{ is below } u_{b_5} \end{cases}$$

We observe that, in all cases, t is a canonical tree model of $\alpha \setminus u$. Let m be a match of α in t. We now show that this leads to a contradiction in all the five cases.

In the cases where u is below u_{b_i} for $i \in \{1, 2, 3, 5\}$, we have $\mathrm{m}(u_{b_i}) = n_{b_i}$. This follows from the relative depths of b-nodes and the fact that u_{a_3} needs to be matched at the only a-node that has an a-child. In all four cases, this leads to a contradiction, as by the definition of $t_{x\setminus u}$, it holds that $t_{x\setminus u} \notin M_t(x)$ and thus this match is not possible.

The remaining case is that u is a descendant of u_{b_4}. The root of α can only be matched to the root of t or one level deeper, as otherwise, the lowest b-nodes, i.e., the nodes u_{b_1} and u_{b_4} cannot be matched anywhere. Therefore, either $\mathrm{m}(u_{b_4})$ or $\mathrm{m}(u_{b_5})$ has to be a node on depth 6. This implies that $\mathrm{m}(u_{b_4}) = n_{b_4}$ or $\mathrm{m}(u_{b_5}) = n_{b_4}$, as n_{b_4} is the only b-node on depth 6. Both matches are not possible, however, as $t_{q\setminus u} \notin M_t(q)$ and $t_{q\setminus u} \notin M_t(r)$. This concludes the proof. □

7.3 Proof of Lemma 6.2

Lemma 6.2: *Let p, q, and r be tree patterns that have non-wildcard roots, do not use labels in $\{a, b\}$, and such that $p \subseteq q \subseteq r$. If*

(1) p and r are minimal, and
(2) there is no tree pattern q' such that
- *$p \subseteq q' \subseteq r$ and*
- *$size(q') < size(q)$,*

then $P(p, q, r)$ is minimal.

Assume that (1) and (2) are true and let $\alpha = P(p, q, r)$. Let β be an arbitrary minimal pattern with $\alpha \equiv \beta$. We will show that β cannot be smaller than α by gradually restricting the structure of β. This proves that α is minimal.

Pattern α is depicted in Figure 16. We annotated α with node names that we will use in the proof. Where possible, we use similar names (with u replaced by v) for corresponding nodes from β, e.g., we use v_ε to denote the root of β.

Recall that β is nonredundant because it is minimal. We now show that α is also nonredundant. By assumption we have that $q \subseteq r$. If there were a leaf n such that $q \setminus n \subseteq r$ then we would have $p \subseteq q \setminus n \subseteq r$, which contradicts (2). Thus, q is barely contained in r and, by Lemma 7.11, α is indeed nonredundant. Therefore we can apply Lemma 7.4 to α and β and obtain that the root of β has exactly two children, v_L and v_R such that

$$\alpha_{u_L} \equiv \beta_{v_L} \text{ and } \alpha_{u_R} \equiv \beta_{v_R}. \tag{$\diamond$}$$

We will prove the following claims, from which we will be able to deduce Lemma 6.2.

(A) β_{v_R} is of the form $* \to * \to * \to * \to a \to b \Rightarrow r'$ with r' equivalent to r.
(B) v_L has a child v_{a_3} such that $\beta^{v_{a_3}} \equiv_S \alpha^{u_{a_3}}$. The tree pattern $\beta^{v_{a_3}}$ has a node v_* and nodes v_{a_i} and v_{b_i} for $i = 1, 2, 3$, all pairwise different. Moreover, $\beta^{v_*} \equiv_S \alpha^{u_*}$ and, for every $i = 1, 2, 3$, we have $\beta^{v_{a_i}} \equiv_S \alpha^{u_{a_i}}$ and $\beta^{v_{b_i}} \equiv_S \alpha^{u_{b_i}}$.
(C) v_L has a child v_{LR} (other than v_{a_3}) that has a descendant on depth 7 in β which is the root of a subpattern q' such that $p \subseteq q' \subseteq r$.

The claims (A), (B), and (C) imply that β is structured like α, with the exceptions that

- the subpatterns below the nodes v_{b_1}, v_{b_2}, v_{b_3}, and v_{b_5} can be different but equivalent;
- 5 levels below v_{LR} there is a subpattern q', possibly different from q, but with $p \subseteq q' \subseteq r$;
- for some edges, we do not know whether they are child edges or descendant edges; and
- β can have additional subpatterns not enforced by the above conditions (attached below nodes on the path from v_L to the subpattern q').

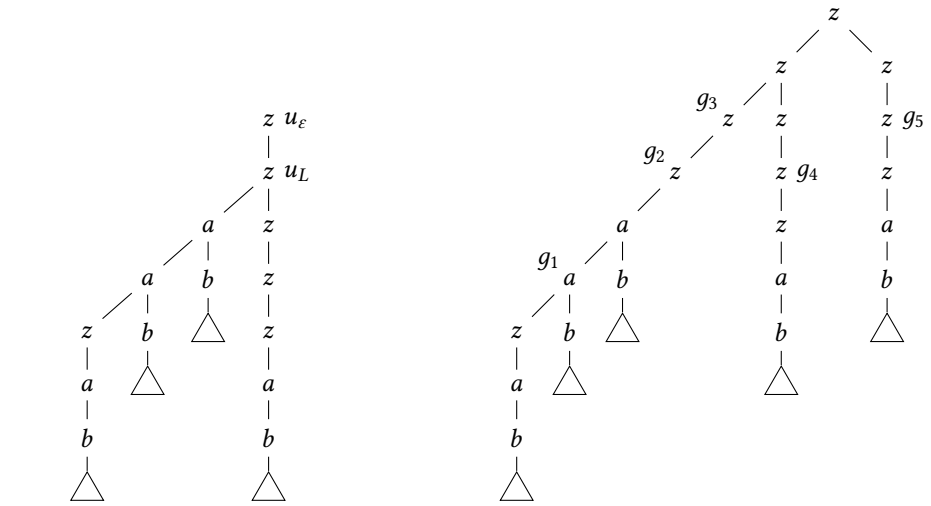
Using conditions (1) and (2) from the lemma statement, we then have that $\text{size}(\alpha) \leq \text{size}(\beta)$. This means that it suffices to prove Claims (A)–(C) to conclude the proof of Lemma 6.2.

It therefore remains to show Claims (A)–(C). Before diving into these proofs, we give a list of some easy "negative" properties, saying that β cannot have particular combinations of nodes as substructures:

(N1) in β_{v_R} there is no a-node with an a-descendant;
(N2) in β there is no b-node with an a-descendant nor with a b-descendant; and
(N3) in β there is no a-node on depth at least 4 whose parent is a-labeled and connected to this node by a child edge;
(N4) in β there is no a-node on depth at least 4 whose grandparent has an a-labeled child, and the edges connecting the grandparent to this node and to the a-labeled child are all child edges;
(N5) in β_{v_L} there is no a-node on depth 4 such that the path from v_L to this node contains only child edges;
(N6) in β there is no a-node on depth 2 such that the path from v_ε to this node contains only child edges.

We now argue that (N1)–(N6) are true. Let us recall that subpatterns p, q, r contain no nodes labeled a or b.

Property (N1) holds because, by ($\diamond$), β_{v_R} has to match in every canonical tree model of α_{u_R}, while in these models there is only one a-node. Similarly, (N2) holds because β has to match in every



(a) Tree t as used in the proof of (N5) (b) Tree t as used in the proof of (N6)

Fig. 18. Trees used in the proof of (N5) and (N6)

canonical tree model of α, while in these models there is no b-node with an a-descendant nor with a b-descendant. For (N3) and (N4), we consider the smallest canonical tree model t_1 of α (i.e., that in which all the descendant edges are instantiated as paths of length one, that is, single edges). Necessarily there is a match of β in t_1. In this match, a hypothetical node of β violating (N3) or (N4) should be matched to an a-node on depth at least 4 such that some child of its grandparent is a-labeled. Such a node does not exist in t_1, so (N3) and (N4) hold.

Suppose that (N5) is violated by some v of β_{v_L}. Consider the smallest canonical tree model t of α_{u_L}, depicted in Figure 18a, and a match m of β_{v_L} in t, which exists by ($\diamond$). Since in t there is no a-node on depth 4 nor on depth higher than 5, necessarily v is matched to a node on depth 5. The node v_L, connected to v on three child edges, has to be matched to a node on depth 2. Then v_ε is matched either to u_L or to u_ε. We may, however, alter m so that it matches v_ε to u_L (it remains a correct match, since v_L is the only child of v_ε in β_{v_L}, and $\mathrm{m}(v_L)$ is a child of u_L). This means that β_{v_L} matches in the subtree t^{u_L} of t, starting in u_L. But then, by ($\diamond$), α_{u_L} has to match in t^{u_L} as well. This is, however, not the case, since α_{u_L} requires an a-node on depth at least 5, while in t^{u_L} there is no such node. This shows that the node v could not exist, and thus (N5) holds.

Finally, we show (N6). Assume towards a contradiction that (N6) is violated by some a-node v on depth 2 in β such that the path from v_ε to v consists of two child edges. Consider the canonical tree model t of α in which (u_L, u_{a_3}) is replaced by a path of length 3 and all the other descendant edges are replaced by child edges. We depict the structure of t in Figure 18b. Let m be a match of β in t. The node v is matched to some a-node of t, and v_ε is matched to the grandparent of $\mathrm{m}(v)$. We have 5 possibilities for $\mathrm{m}(v_\varepsilon)$, denoted as g_1–g_5. In any case, β matches in the subtree t^{g_i}, and thus also α, which is equivalent to β, matches in this subtree. We can see, however, that α does not match t^{g_i} for any $i \in \{1, 2, 3, 4, 5\}$. Thus our assumption was wrong, which finishes the proof of (N6).

We now come back to the proof of claims (A)–(C).

Claim (A): *Pattern β_{v_R} is of the form*

$$* \longrightarrow * \longrightarrow * \longrightarrow * \longrightarrow a \longrightarrow b \Rightarrow r'$$

with r' equivalent to r.

Proof. We first prove that there is a node v_{a_5} on depth 4 in β_{v_R} such that $\alpha^{u_{a_5}} \equiv_S \beta^{v_{a_5}}$ and on the path from v_ε to v_{a_5} there are only child edges. To this end, consider a canonical tree model t_{v_L} of β_{v_L} such that β does not match in t_{v_L} (it exists because β is nonredundant, and thus $\beta_{v_L} \not\subseteq \beta$). Consider also a canonical tree model t_{v_R} of β_{v_R} constructed as follows:

- in every a-node v such that $\beta^v \not\subseteq_S \alpha^{u_{a_5}}$ we attach a canonical tree model t^v of β^v such that $\alpha^{u_{a_5}}$ does not match strongly in t^v (it is possible to do this independently for every such a-node thanks to (N1)); this fixes lengths of paths instantiated for some descendant edges;
- all other descendant edges of β_{v_R} (in particular all those that are above a-labeled nodes) are instantiated as paths of length 5.

Let t be the tree obtained from t_{v_L} and t_{v_R} by merging their roots. We have $t \in \mathrm{Can}(\beta)$.

Since $\alpha \equiv \beta$, there is a match m of α in t. By assumption, $t_{v_L} \notin M_t(\beta) = M_t(\alpha)$. Moreover, $t_{v_R} \notin M_t(\alpha_{u_L})$ because in t_{v_R} there is no a-node with an a-descendant by (N1). Thus m has to match u_ε to the root of t, nodes of α_{u_L} to nodes of t_{v_L}, and nodes of α_{u_R} to nodes of t_{v_R}. In particular, u_{a_5} is matched to an a-node v_{a_5} on depth 4 in t_{v_R}. Since v_{a_5} is a labeled node, it is a node that is also in β. We necessarily have $\beta^{v_{a_5}} \subseteq_S \alpha^{u_{a_5}}$, because all subtrees t^v starting in a-nodes v with $\beta^v \not\subseteq_S \alpha^{u_{a_5}}$ are by construction such that $\alpha^{u_{a_5}}$ does not match strongly in t^v. Moreover, there is no descendant edge on the path from v_ε to v_{a_5} in β, as it would have been replaced by a path of length 5 in t, while v_{a_5} is on depth 4 in t. This shows that v_{a_5} is a node on depth 4 in β_{v_R}, that there are only child edges on the path from v_ε to v_{a_5}, and that $\beta^{v_{a_5}} \subseteq_S \alpha^{v_{a_5}}$.

Since $\alpha_{u_R} \subseteq \beta_{v_R}$ by ($\diamond$), by Lemma 7.9 there has to be a node u on depth 4 in α_{u_R} such that $\alpha^u \subseteq \beta^{v_{a_5}}$. As α_{u_R} only has one node on depth 4, we have that $u = u_{a_5}$. We therefore established that $\alpha^{u_{a_5}} \equiv \beta^{v_{a_5}}$ and, by Lemma 7.2, $\alpha^{u_{a_5}} \equiv_S \beta^{v_{a_5}}$.

From the above we can deduce that $\alpha_{u_R} \equiv_S \beta_{v_R}$. Indeed, consider a canonical tree model t of α_{u_R}. Because $\alpha_{u_R} \equiv \beta_{v_R}$ there is a match of β_{v_R} in t. In t there is only one a-node, on depth 4, and thus v_{a_5} has to be matched to this node. It follows that the root of β_{v_R} is matched to the root of t, and hence this is a strong match. Conversely, consider a canonical tree model t of β_{v_R}. Then, by $\alpha^{u_{a_5}} \equiv_S \beta^{v_{a_5}}$, there is a strong match of $\alpha^{u_{a_5}}$ in the subtree $t^{v_{a_5}}$. It can be extended to a strong match of the whole α_{u_R} in t, because v_{a_5} is on depth 4 in t. This shows that $\alpha_{u_R} \equiv_S \beta_{v_R}$.

We obtain (A) by repeatedly applying Lemmas 7.3, 7.5, and 7.8 to subpatterns of α_{u_R} and β_{v_R} starting in nodes on depths from 0 to 5. We start by applying Lemma 7.3 to α_{u_R} and β_{v_R} and deduce that u_ε in α and v_ε in β have the same label. Then, we apply Lemma 7.8(b) to these patterns, and obtain that $\alpha^{u_R} \equiv_S \beta^{v_R}$. On the next level, we additionally use Lemma 7.5 to ensure that v_R has only one child (this lemma requires the patterns to be non-strongly-redundant, but recall that α and β are nonredundant, which implies that all their subpatterns are non-strongly-redundant), and we may again apply Lemmas 7.3 and 7.8(b). On depths 4 and 5 we use case (a) of Lemma 7.8, which additionally ensures that the edges starting in v_{a_5} and in its child are a child edge and a descendant edge, respectively. On depth 5 we use the assumption that the root of r is not a wildcard, in which case we use Lemma 7.8(a). This concludes the proof of (A). □

We now prove claim (B).

Claim (B): *v_L has a child v_{a_3} such that $\beta^{v_{a_3}} \equiv_S \alpha^{u_{a_3}}$. The tree pattern $\beta^{v_{a_3}}$ has a node v_* and nodes v_{a_i} and v_{b_i} for $i = 1, 2, 3$, all pairwise different. Moreover, $\beta^{v_*} \equiv_S \alpha^{u_*}$ and, for every $i = 1, 2, 3$, we have $\beta^{v_{a_i}} \equiv_S \alpha^{u_{a_i}}$ and $\beta^{v_{b_i}} \equiv_S \alpha^{u_{b_i}}$.*

Proof. Essentially, we want to show that v_L has a child v_{a_3} such that $\beta^{v_{a_3}}$ is structured like $\alpha^{u_{a_3}}$, except that the subpatterns rooted below b-nodes may be different but equivalent.

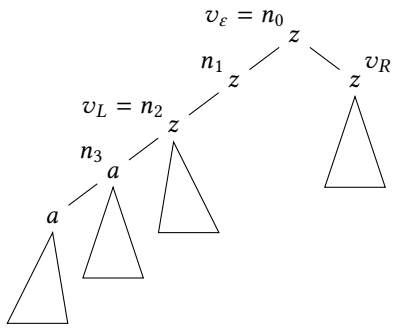


Fig. 19. Sketch of the tree t, as used in the proof of (C1)

By ($\diamond$) we know that $\alpha_{u_L} \equiv \beta_{v_L}$. By Lemma 7.9, there exists a node v_{a_3} on depth 2 in β_{v_L} such that $\beta^{v_{a_3}} \subseteq \alpha^{u_{a_3}}$. As v_ε has only one child in β_{v_L}, this means that v_{a_3} has to be a child of v_L.

Using Lemma 7.9, once more, we also get that there is a node u on depth 2 in α_{u_L} such that $\alpha^u \subseteq \beta^{v_{a_3}} \subseteq \alpha^{u_{a_3}}$. The only possibility is that $u = u_{a_3}$, because $\alpha^{u_{LR}} \not\subseteq \alpha^{u_{a_3}}$. As $\alpha^{u_{a_3}} \equiv \beta^{v_{a_3}}$ and u_{a_3} is not labeled by a wildcard, we obtain by Lemma 7.2 that $\alpha^{u_{a_3}} \equiv_S \beta^{v_{a_3}}$.

As in (A), we can now derive (B) alternatingly using Lemmas 7.3, 7.5, and 7.8, starting at u_{a_3} and v_{a_3}. The first application of Lemma 7.5 yields that v_{a_3} has two children v_{a_2} and v_{b_3}, such that $(\alpha^{u_{a_3}})_{u_{a_2}} \equiv_S (\beta^{v_{a_3}})_{v_{a_2}}$ and $(\alpha^{u_{a_3}})_{u_{b_3}} \equiv_S (\beta^{v_{a_3}})_{v_{b_3}}$, while Lemma 7.3 ensures that v_{a_3} has label a. Using Lemma 7.8(a) twice, we get $\beta^{v_{a_2}} \equiv_S \alpha^{u_{a_2}}$ and $\beta^{v_{b_3}} \equiv_S \alpha^{u_{b_3}}$, and that the edges leading to v_{a_2} and v_{b_3} are child edges. Continuing like this finally yields (B). □

Claim (C): *Node v_L in β has a child v_{LR} (different from v_{a_3}) that has a descendant on depth 7 which is the root of a subpattern q' such that $p \subseteq q' \subseteq r$.*

Proof. We first prove the following properties, which we will need to prove the claim:

(C1) the label of v_L is $*$, (v_ε, v_L) is a child edge, and (v_L, v_{a_3}) is a descendant edge; and
(C2) in β there is only one a-node (namely v_{a_3}) that has an a-child attached on a child edge.

We first prove (C1). Since $\alpha_{u_L} \equiv \beta_{v_L}$, we obtain $\alpha^{u_L} \equiv \beta^{v_L}$ from Lemma 7.7. Thus, by Lemma 7.3, v_L has label $*$.

Let us prove the part that (v_ε, v_L) is a child edge. Towards a contradiction, assume that (v_ε, v_L) is a descendant edge. Then consider the canonical tree model $t \in \mathrm{Can}(\beta)$ such that

- (v_ε, v_L) is instantiated as a path of length two,
- every descendant edge that starts in v_L and leads to an a-labeled child having itself an a-labeled child connected to it by a child edge (in particular the edge (v_L, v_{a_3}), if this is at all a descendant edge) is replaced by a single edge, and
- all other descendant edges are replaced by paths of length 4.

As previously, we identify nodes of β with corresponding nodes of t. We have sketched t in Figure 19. We will show that α does not match t, which will contradict the fact that $\alpha \equiv \beta$. Notice that u_{a_3} of α needs to be matched to some child of v_L in t, as by (N3) (and because we have replaced descendant edges of β by sequences of edges in t) it cannot be matched anywhere deeper, and by Claim (A) it cannot be matched in t_{v_R}. Thus assume that u_{a_3} is matched to a node on depth 3 in t. We name this node n_3 and its ancestors on depth 0, 1, and 2 we name $n_0 = v_\varepsilon$, n_1, and $n_2 = v_L$. We now have two cases. The nodes u_ε and u_L of α are either mapped to n_0 and n_1, respectively, or to n_1 and n_2, respectively. We show that both cases are impossible.

In the case where u_ε is mapped to n_0, the node u_{a_4} has to be mapped to some a-labeled node on depth 5. In the case where u_ε is mapped to n_1, the node u_{a_5} has to be mapped to some a-labeled node on depth 5. We will show, however, that there is no a-labeled node on depth 5 in t, ruling out both cases. Clearly, the subtree t^{v_R} does not have such a node by Claim (A).

We now consider the other subtree, i.e., the one corresponding to β^{v_L}. Node v_L is on depth 2 in t. Consider an arbitrary a-labeled descendant n_a of v_L in t; it is simultaneously a node of β. If n_a is in $t^{v_{a_3}}$ we know that it is on depth 3, 4, or 6, since v_{a_3} is on depth 3 in t and Claim (B) holds. If n_a is not in $t^{v_{a_3}}$, we consider three cases.

- If in β the node v_L is connected to n_a only through child edges, n_a does not have depth 5 (i.e., depth 4 in β) by (N5).
- If in β some edge between v_L and n_a is a descendant edge that was replaced in t by a path of length 4, the depth of n_a in t is at least 6=2+4.
- In the remaining case, the path from v_L to n_a consists of a descendant edge followed by child edges, where the node n' after the descendant edge is a-labeled and has an a-labeled child connected to n' by a child edge. Then n_a cannot be on depth 5 in t due to (N4) (as then it would be two levels below n').

Thus there is indeed no a-labeled node on depth 5 in t. Therefore the edge (v_ε, v_L) has to be a child edge. Then (v_L, v_{a_3}) has to be a descendant edge due to (N6). This shows (C1).

We now prove that (C2) holds. To the contrary, suppose that there is an a-node v that has an a-child attached on a child edge, and such that $v \neq v_{a_3}$. By (N3) v is on depth at most 2, by Claim (A) it is not in β_{v_R}, and by (C1) we have $v \neq v_L$. Thus v is a child of v_L. Moreover, by (N6), we have that (v_L, v) is a descendant edge.

We will show that $\alpha^{u_{a_3}} \subseteq \beta^v$. To this end, take any canonical tree model $t^{u_{a_3}}$ of $\alpha^{u_{a_3}}$, and extend it to a tree model t of α in any way. Because $\alpha \equiv \beta$, we have a match m of β in t. In t there is only one a-node with an a-child, namely u_{a_3}, and thus v has to be matched to u_{a_3}. This means that the part of m concerning only nodes of β^v is a match of β^v in $t^{u_{a_3}}$. As this can be done for every canonical tree model of $\alpha^{u_{a_3}}$, we obtain $\alpha^{u_{a_3}} \subseteq \beta^v$.

By Claim (B) we have $\beta^{v_{a_3}} \equiv \alpha^{u_{a_3}}$, and thus actually $\beta^{v_{a_3}} \subseteq \beta^v$. Since v is attached to its parent on a descendant edge, it should be clear that $\beta \equiv \beta \setminus v$, and thus β is redundant, contrary to our assumption. It follows that the considered node v could not exist, which finishes the proof of (C2).

We now continue with proving Claim (C), i.e., we show that in β, the node v_L has a child v_{LR} (other than v_{a_3}) that has a descendant on depth 7 which is the root of a subpattern q' such that $p \subseteq q' \subseteq r$.

To this end, consider a canonical tree model $t \in \mathrm{Can}(\beta)$ such that

- (v_L, v_{a_3}) is instantiated as a path of length two,
- in every b-node v such that $\beta^v \not\subseteq \alpha^{u_{b_5}}$ we attach a canonical tree model t^v of β^v such that $\alpha^{u_{b_5}}$ does not match in t^v (it is possible to do this independently for every such b-node thanks to (N2)); this fixes lengths of paths instantiated for some descendant edges;
- all other descendant edges of β are instantiated as paths of length 6.

As always, we identify nodes of β with the corresponding nodes in t. We sketched t in Figure 20. Just ignore the subtree marked (‡) for now. It will become important later.

We first show that if an a-node v_a in t has an a-child v'_a, then $v_a = v_{a_3}$. Indeed, by (N2), v_a has no b-ancestor. Thus, by construction of t, every descendant edge starting in v_a is replaced by a path of length at least two. It follows that the edge (v_a, v'_a) was a child edge in β. Thus by (C2) we necessarily have $v_a = v_{a_3}$.

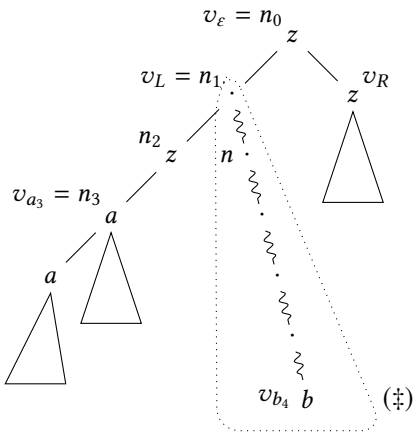


Fig. 20. Sketch of a tree t, as used in the proof of (C)

As $\beta \subseteq \alpha$, there is a match m of α in t. Surely u_{a_3} is matched to an a-node that has an a-child in t. Thus, by the above, $\mathrm{m}(u_{a_3}) = v_{a_3}$. We name this node n_3 and its ancestors on depth i, we call n_i, similarly as in the proof of (C1). Nodes u_ε and u_L have to be matched either to n_0 and n_1, respectively (Case 1), or to n_1 and n_2, respectively (Case 2).

Let us first consider Case 1. Then u_L is matched to v_L. The node u_{b_4} has to be matched to a b-labeled node 5 levels below v_L. Consider now Case 2, where u_ε is matched to v_L. Then the node u_{b_5} has to be matched to a b-labeled node 5 levels below v_L. So, irrespective of Case 1 or 2, we know that there must be a b-labeled node 5 levels below v_L.

Clearly in t^{n_3}—the image of the subpattern $\beta^{v_{a_3}}$—there is no such node due to Claim (B). Therefore, in t there has to be some other child n of n_1 and 4 levels below n, there has to be a b-node v_{b_4}. We sketched this subtree as (‡) in Figure 20. Because, by (N2), v_{b_4} has no b-ancestor, by construction of t we notice that every descendant edge between v_L and v_{b_4} in β is replaced in t by a path of length 6. Thus, because v_{b_4} is only 5 levels below v_L in t, we we can furthermore conclude that v_L is connected to v_{v_4} using only child edges, and hence it is on depth 6 in β.

We already know that m either matches the subpattern $\alpha^{u_{b_4}}$ in the subtree $t^{v_{b_4}}$ (Case 1) or matches the subpattern $\alpha^{u_{b_5}}$ in the subtree $t^{v_{b_4}}$ (Case 2). Since $q \subseteq r$, we have that $\alpha^{u_{b_4}} \subseteq \alpha^{u_{b_5}}$. We can conclude that in both cases it is possible to match $\alpha^{u_{b_5}}$ in $t^{v_{b_4}}$. We necessarily have $\beta^{v_{b_4}} \subseteq \alpha^{u_{a_5}}$, because all subtrees t^v starting in b-nodes v with $\beta^v \not\subseteq \alpha^{u_{a_5}}$ are by construction such that $\alpha^{u_{a_5}}$ does not match in t^v.

Now we aim at showing $\alpha^{u_{b_1}} \subseteq \beta^{v_{b_4}}$. We know by Lemma 7.9 that there has to be a node u on depth 6 in α such that $\alpha^u \subseteq \beta^{v_{b_4}}$. As there are only two b-labeled nodes on depth 6 in α, we either have that $u = u_{b_4}$ or $u = u_{b_1}$. As furthermore $\alpha^{u_{b_1}} \subseteq \alpha^{u_{b_4}}$, we can conclude that $\alpha^{u_{b_1}} \subseteq \beta^{v_{b_4}}$.

Summarizing, we have that $\alpha^{u_{b_1}} \subseteq \beta^{v_{b_4}} \subseteq \alpha^{u_{b_5}}$. By Lemma 7.9 applied to the child of u_{b_5}, we obtain that v_{b_4} has a child v such that $\beta^v \subseteq r$. Moreover, by Lemma 7.9 applied to v, we obtain that u_{b_1} has a child u such that $\alpha^u \subseteq \beta^v$. As u_{b_1} has only one child, which is the root of p, we can conclude that $p \subseteq \beta^v \subseteq r$. This finishes the proof of Claim (C). □

8 DISCUSSION AND OUTLOOK

In Table 1 we give an overview the complexities of tree pattern optimization problems from [29] and this work. We provided several new insights on the minimization problems for tree patterns and generalized tree patterns. It is the first to give a tight complexity bound for these problems. From

Problem	Containment		Equivalence		Minimization	
Queries	TP	GTP	TP	GTP	TP	GTP
On trees	coNP [a]	coNP [b]	coNP [a]	coNP [c]	Σ_2^P [e]	Σ_2^P [f]
On graphs	coNP [a]	coNP [b,d]	coNP [a]	coNP [c,d]	Σ_2^P [e,d]	Σ_2^P [f,d]

[a] Miklau and Suciu [29] [b] Proposition 2.5 [c] Corollary 2.4 [d] Proposition 2.6
[e] Theorem 3.1 [f] Theorem 3.2

Table 1. A complexity overview of containment, equivalence, and minimization for tree patterns (TP) and generalized tree patterns (GTP). All complexities are completeness results.

Section 5.1 we now know that such patterns cannot be, in general, minimized by removing nodes. In contrast, for tree patterns, it is known that minimization by removing nodes is possible if the patterns are $*$-narrow (Lemma 2.9, cfr. [20]). It would be interesting to see if $*$-narrow generalized tree patterns can also be minimized by removing nodes.

From the example in Section 5.1, we know that it may also be necessary to merge nodes to reach a minimal pattern from a given one. It is now natural to ask if *merging and deleting nodes* always suffices for minimizing patterns. We first observe that, if patterns can be minimized by merging and deleting nodes, then the order in which these operations occur is not important. Indeed, assume that we are given a (non-minimal) pattern p and assume that we obtained an equivalent pattern p_1 from p by deleting a node or merging two nodes. If patterns can always be minimized by merging and deleting nodes, then we can also go from p_1 to a minimal pattern (which would also be equivalent to p) by merging and deleting nodes.

Given our main complexity results (Theorems 3.2 and 3.1), the existence of an algorithm that minimizes patterns in this way seems unlikely. Indeed, the theorems imply that that, if one can minimize patterns by merging and removing nodes, then $\Sigma_2^P = \text{coNP}$. Following the idea for the NP upper bound from [19, Lemma 6], one would be able to decide the complement of minimization by guessing nodes that cannot be deleted or merged, plus guessing trees witnessing why deleting or merging these nodes would result in non-equivalent patterns.

We believe that the above argument can be strengthened so that the $\Sigma_2^P \neq \text{coNP}$ condition is not needed. This proof would be based on the concrete patterns we construct in the proof of Lemma 6.5. Essentially, the argument boils down to showing that there are patterns q'' with $p \subseteq q'' \subseteq r$ from which no smallest pattern in $\{q' \mid p \subseteq q' \subseteq r\}$ 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 $\{q' \mid p \subseteq q' \subseteq r\}$. So, a rewriting sequence from a given pattern to an equivalent minimal one may have to perform steps that make patterns larger.

Operations for Reaching Minimal Patterns. 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 2009 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 et al. 2010, 2011 considered this question for tree patterns and provided a set of axioms complete up to homomorphism containment.

In the light of what we have seen in this paper, we believe that, by allowing (1) deletion of nodes, (2) merging of nodes, and (3) splitting of nodes it may be possible to reach a minimal equivalent pattern from each given pattern. These operations suffice for all patterns we have considered in this paper but we do not know if they are sufficient in general.

Reflexive transitive closure. A final but very interesting question is to which extent the results of this paper can be strengthened to queries that have *reflexive* transitive closure. Reflexive transitive closure is interesting in the context of querying graphs and, for many queries, more natural to ask for than transitive closure. However, our proofs do not naively carry over to queries with reflexive transitive closure and it would be interesting to see if they can be adapted.

ACKNOWLEDGMENTS

We are very grateful to Benny Kimelfeld for insightful discussions and for bringing the tree pattern query minimization problem to our attention. We thank the anonymous reviewers of J. ACM for many helpful comments.

This article is the full version of [14], which was presented at the 35th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS) in San Francisco, California, USA, 2016. We acknowledge the support of Poland's National Science Centre grant no. UMO-2013/11/D/ST6/03075 and grant number MA 4938/2–1 from the Deutsche Forschungsgemeinschaft.

REFERENCES

[1] Serge Abiteboul, Luc Segoufin, and Victor Vianu. 2009. Static Analysis of Active XML Systems. *ACM Trans. Database Syst.* 34, 4 (2009).

[2] Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan, and Divesh Srivastava. 2002. Tree Pattern Query Minimization. *VLDB J.* 11, 4 (2002), 315–331.

[3] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. 2016. Foundations of Modern Graph Query Languages. *CoRR* abs/1610.06264 (2016).

[4] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. 2014. *Foundations of Data Exchange.* Cambridge University Press.

[5] Marcelo Arenas, Sebastián Conca, and Jorge Pérez. 2012. Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths will Prevent Adoption of the Standard. In *World Wide Web Conference (WWW).* 629–638.

[6] Marcelo Arenas and Leonid Libkin. 2008. XML Data Exchange: Consistency and Query Answering. *J. ACM* 58, 1 (2008).

[7] Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. 2010. XML with Incomplete Information. *J. ACM* 58, 1 (2010), 4.

[8] Michael Benedikt, Wenfei Fan, and Floris Geerts. 2008. XPath Satisfiability in the Presence of DTDs. *J. ACM* 55, 2 (2008).

[9] Henrik Björklund, Wim Martens, and Thomas Schwentick. 2011. Conjunctive Query Containment over Trees. *J. Comput. Syst. Sci.* 77, 3 (2011), 450–472.

[10] Henrik Björklund, Wim Martens, and Thomas Schwentick. 2013. Validity of Tree Pattern Queries with Respect to Schema Information. In *International Symposium on Mathematical Foundations of Computer Science (MFCS).* 171–182.

[11] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An Analytical Study of Large SPARQL Query Logs. *CoRR* abs/1708.00363 (2017). To appear in VLDB 2018.

[12] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Symposium on Theory of Computing (STOC).* 77–90.

[13] Ding Chen and Chee Yong Chan. 2008. Minimization of Tree Pattern Queries with Constraints. In *International Conference on Management of Data (SIGMOD).* 609–622.

[14] Wojciech Czerwiński, Wim Martens, Matthias Niewerth, and Paweł Parys. 2016. Minimization of Tree Pattern Queries. In *Symposium on Principles of Database Systems (PODS).* 43–54.

[15] Wojciech Czerwiński, Wim Martens, Paweł Parys, and Marcin Przybyłko. 2015. The (Almost) Complete Guide to Tree Pattern Containment. In *Symposium on Principles of Database Systems (PODS).* 117–130.

[16] Claire David, Amélie Gheerbrant, Leonid Libkin, and Wim Martens. 2013. Containment of Pattern-Based Queries over Data Trees. In *International Conference on Database Theory (ICDT).* 201–212.

[17] Bettina Fazzinga, Sergio Flesca, and Filippo Furfaro. 2010. On the Expressiveness of Generalization Rules for XPath Query Relaxation. In *International Database Engineering and Applications Symposium (IDEAS)*. 157–168.
[18] Bettina Fazzinga, Sergio Flesca, and Filippo Furfaro. 2011. XPath Query Relaxation through Rewriting Rules. *IEEE Trans. Knowl. Data Eng.* 23, 10 (2011), 1583–1600.
[19] Sergio Flesca, Filippo Furfaro, and Elio Masciari. 2003. On the Minimization of XPath Queries. In *International Conference on Very Large Data Bases (VLDB)*. 153–164.
[20] Sergio Flesca, Filippo Furfaro, and Elio Masciari. 2008. On the Minimization of XPath Queries. *J. ACM* 55, 1 (2008).
[21] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. 2013. Reasoning About Pattern-Based XML Queries. In *International Conference on Web Reasoning and Rule Systems (RR)*. 4–18.
[22] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree Decompositions: Questions and Answers. In *Symposium on Principles of Database Systems (PODS)*. 57–74.
[23] Georg Gottlob, Christoph Koch, and Klaus U. Schulz. 2006. Conjunctive Queries over Trees. *J. ACM* 53, 2 (2006), 238–272.
[24] Gremlin 2013. Gremlin Language. https://github.com/tinkerpop/gremlin/wiki. (2013).
[25] JSONPath 2016. http://jsonpath.com/. (2016).
[26] Benny Kimelfeld and Yehoshua Sagiv. 2008. Revisiting Redundancy and Minimization in an XPath Fragment. In *International Conference on Extending Database Technology (EDBT)*. 61–72.
[27] Leonid Libkin, Wim Martens, and Domagoj Vrgoc. 2016. Querying Graphs with Data. *J. ACM* 63, 2 (2016), 14.
[28] Katja Losemann and Wim Martens. 2013. The Complexity of Regular Expressions and Property Paths in SPARQL. *ACM Trans. Database Syst.* 38, 4 (2013), 24.
[29] Gerome Miklau and Dan Suciu. 2004. Containment and Equivalence for a Fragment of XPath. *J. ACM* 51, 1 (2004), 2–45.
[30] Tova Milo and Dan Suciu. 1999. Index Structures for Path Expressions. In *International Conference on Database Theory (ICDT)*. 277–295.
[31] Neo4j Cypher 2016. The Cypher Query Language. https://neo4j.com/docs/developer-manual/current/cypher/. (2016).
[32] Frank Neven and Thomas Schwentick. 2006. On the Complexity of XPath Containment in the Presence of Disjunction, DTDs, and Variables. *Logical Methods in Computer Science* 2, 3 (2006).
[33] Prakash Ramanan. 2002. Efficient Algorithms for Minimizing Tree Pattern Queries. In *International Conference on Management of Data (SIGMOD)*. 299–309.
[34] Jonathan Robie, Don Chamberlin, Michael Dyck, and John Snelson. 2014. *XML Path Language 3.0*. Technical Report. World Wide Web Consortium. Recommendation, http://www.w3.org/TR/2014/REC-xpath-30-20140408/.
[35] Ian Robinson, Jim Webber, and Emil Eifrem. 2015. *Graph Databases* (2 ed.). O'Reilly.
[36] Marko A. Rodriguez. 2015. The Gremlin Graph Traversal Machine and Language (Invited Talk). In *Symposium on Database Programming Languages (DBPL)*. 1–10.
[37] Slawek Staworko and Piotr Wieczorek. 2015. Characterizing XML Twig Queries with Examples. In *International Conference on Database Theory (ICDT)*. 144–160.
[38] Larry J. Stockmeyer. 1976. The Polynomial-Time Hierarchy. *Theor. Comput. Sci.* 3, 1 (1976), 1–22.
[39] Balder ten Cate and Maarten Marx. 2009. Axiomatizing the Logical Core of XPath 2.0. *Theory Comput. Syst.* 44, 4 (2009), 561–589.
[40] W3C Sparql 2013. SPARQL 1.1 Query Language. https://www.w3.org/TR/sparql11-query/. (2013). World Wide Web Consortium.
[41] Wikidata Examples 2016. WikiData SPARQL Query Service Examples. https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples. (2016).
[42] Peter T. Wood. 2001. Minimising Simple XPath Expressions. In *WebDB*. 13–18.
[43] Wanhong Xu and Z. Meral Özsoyoglu. 2005. Rewriting XPath Queries Using Materialized Views. In *International Conference on Very Large Data Bases (VLDB)*. 121–132.

A APPENDIX

Proposition 2.4. *Let p_1 and p_2 be generalized tree patterns. Then*

$$p_1 \subseteq p_2 \text{ if and only if } M_t(p_1) \subseteq M_t(p_2) \ .$$

The same holds if p_1 and p_2 are tree patterns.

Proof. The following proof is a minor adaptation of the argument in [29, Section 5.3]. Let unfold(G) be the (possibly infinite) tree unfolding of some graph G, where edge labels are preserved. Then, for any pattern p, the following are equivalent: (1) p can be matched in G, (2) p can be matched in unfold(G), (3) there exists a finite subtree t of unfold(G) such that p can be matched in t.

Thus, if there exists a witness graph G such that p_1 can be matched in G but p_2 cannot, then we can construct a witness tree t such that p_1 can be matched in t and p_2 cannot: just take the finite subree t of unfold(G) as above. Notice that p_2 cannot be matched in t, since any embedding from p_2 to t extends to an embedding from p_2 to G. This shows that $M_t(p_1) \subseteq M_t(p_2)$ implies that $p_1 \subseteq p_2$. The other direction is trivial. □

Proposition 2.5. *Generalized Tree Pattern Containment is* coNP*-complete.*

Proof. The coNP lower bound is immediate from [29, Theorem 4]. It therefore suffices to prove the upper bound. We use some ideas from [29].

Let p_1 and p_2 be two generalized tree patterns. For a tree t we define its size, denoted size(t), as the number of its nodes. We show the upper bound by a small model property, i.e., we will show that

(SMP) If $M_t(p_1) \setminus M_t(p_2)$ is nonempty, then there is a tree in $M_t(p_1) \setminus M_t(p_2)$ of size at most $2 \cdot \text{size}(p_1) \cdot \text{size}(p_2)$.

The coNP upper bound follows from (SMP) by the trivial algorithm that guesses a counterexample t of size at most $2 \cdot \text{size}(p_1) \cdot \text{size}(p_2)$ and then verifies in polynomial time that $t \in M_t(p_1) \setminus M_t(p_2)$. By Proposition 2.4, this yields the statement.

It remains to show (SMP). Let t be a smallest tree in $M_t(p_1) \setminus M_t(p_2)$ and let m_1 be a match of p_1 in t. We show that

(a) for every node v that does not appear in the image of m_1,
- a descendant of v appears in the image of m_1; and
- an ancestor of v appears in the image of m_1;

and, furthermore,

(b) for every path $\pi = v_1 \cdots v_n$ in t with $n \geq \text{size}(p_2)$ and such that every node of π has exactly one child, some node of π appears in the image of m_1.

From (a) and (b), we can conclude that $\text{size}(t) \leq 2 \cdot \text{size}(p_1) \cdot \text{size}(p_2)$, as in t there can be only the following nodes:

- up to size(p_1) nodes that appear in the image of m_1;
- up to size(p_1) nodes that have more than one child (since t has at most as many leaves as p_1); and
- at most $2 \cdot \text{size}(p_1) - 1$ many maximal paths strictly between these nodes, each of them consisting of at most $\text{size}(p_2) - 1$ nodes, due to (b).

It remains to show (a) and (b). Assume towards a contradiction that (a) is not satisfied. We consider two cases. First, if a node v and none of its descendants appear in the image of m_1, then we can remove the subtree rooted at v from t. The resulting tree would also not be in $M_t(p_2)$. However, it would still be in $M_t(p_1)$ because m_1 is still a match. This contradicts the assumption that t is as small as possible in $M_t(p_1) \setminus M_t(p_2)$. Second, if a node v and none of its ancestors appear in the

image of m_1, then we can remove all nodes from t that are not descendants of $m_1(u)$, where u is the root of p_1. Again, the resulting tree is smaller than t (because v exists) and is still in $M_t(p_1) \setminus M_t(p_2)$, which is again a contradiction.

We now show (b). Towards a contradiction, assume that $\pi = v_1 \cdots v_n$ is a path in t with $n \geq \text{size}(p_2)$, every node of π has exactly one child, and no node of π is in the image of m_1. Let v_{n+1} be the only child of v_n. Again, we will construct a smaller tree $t' \in M_t(p_1) \setminus M_t(p_2)$. We construct t' from t by removing all the nodes $v_{\text{size}(p_2)}, \ldots, v_n$ and changing the only child of $v_{\text{size}(p_2)-1}$ to be v_{n+1}. Furthermore, we change the node labels of $v_1, \ldots, v_{\text{size}(p_2)-1}$ to z, where z is a label that does not appear in p_2. If the labels on the edges $(v_1, v_2), \ldots, (v_n, v_{n+1})$ are not identical, we know that they cannot witness a labeled transitive-closure test in p_1, and, in this case, we change all edge labels on $(v_1, v_2), \ldots, (v_{\text{size}(p_2)-1}, v_{n+1})$ to z. Otherwise all edges (including the new edge $(v_{\text{size}(p_2)-1}, v_{n+1})$) keep the common label.

Notice that we have that $t' \in M_t(p_1)$ because m_1 is still a match. It remains to show that $t' \notin M_t(p_2)$. Assume otherwise and let m_2' be a match of p_2 in t'. We will show that in this case $t \in M_t(p_2)$, contradicting the assumption that $t \in M_t(p_1) \setminus M_t(p_2)$.

Let π' be the path $v_1 \cdots v_{\text{size}(p_2)-1}$ in t'. We will construct a match m_2 of p_2 in t. The match m_2 is identical to m_2' for all nodes w of p_2 for which

- $m_2'(w) \notin \pi'$; or
- w is connected by a path consisting only of child edges to an ancestor u of w such that $m_2'(u) \notin \pi'$.

For all other nodes w, we define $m_2(w) = v_{j+n-\text{size}(p_2)+1}$, where j is such that $m_2'(w) = v_j$. We now show that m_2 is indeed a match, by proving that conditions (1)–(3) in the definition of a match are fulfilled.

Condition (1) carries over from m_2' to m_2, because all nodes w with $\text{lab}(m_2'(w)) \neq \text{lab}(m_2(w))$ have $\text{lab}(m_2'(w)) = z$ and thus $\text{lab}(w) = *$.

Condition (2) carries over by a case analysis on the match of the endpoints of simple edges (u, w) of p_2. In the following cases, (2) holds because the match of the edge is not changed:

- $m_2'(u) \notin \pi'$ and $m_2'(w) \notin \pi'$: The edge is matched outside of π' by m_2' and thus matched outside of π in m_2.
- $m_2'(w) = v_1$: By definition, $m_2(u) = m_2'(u)$ and $m_2(w) = m_2'(w)$.
- $m_2'(u) = v_{\text{size}(p_2)-1}$: In this case it is not possible that u is connected to some ancestor u' of u (with $m_2'(u') \notin \pi'$) only by child edges, because of the length of π' and the size of p_2. Therefore, $m_2(u) = v_n$, as $\text{size}(p_2) - 1 + n - \text{size}(p_2) + 1 = n$. We note that $m_2(w) = m_2'(w)$ is the only child of v_n.
- $m_2'(u) \in \pi'$ and $m_2'(w) \in \pi'$: Here, either both nodes are mapped identically as in m_2', or both nodes are mapped $n - \text{size}(p_2) + 1$ levels deeper than in m_2'. Therefore $(m_2(u), m_2(w))$ is an edge in t.

For the labels, we have the same observation as in the case of nodes. If for some edge (u, w) of p_2, we have that $(m_2'(u), m_2'(w)) \neq (m_2(u), m_2(w))$, then $\text{lab}(u, w) = *$.

Condition (3) follows by similar arguments. We note that the path from $m_2(u)$ to $m_2(w)$ can be $n - \text{size}(p_2) + 1$ nodes longer than the path from $m_2'(u)$ to $m_2'(w)$ in some cases, for example, when $m_2'(u)$ is above v_1 and $m_2'(w)$ below $v_{\text{size}(p_2)-1}$. □