Bag Equivalence of Tree Patterns

Sara Cohen and Yaacov Y. Weiss

Abstract

When a query is evaluated under bag semantics, each answer is returned as many times as it has derivations. Bag semantics has long been recognized as important, especially when aggregation functions will be applied to query results. This article is the first to focus on bag semantics for tree pattern queries. In particular, the problem of bag-equivalence of a large class of tree pattern queries (which can be used to model XPath) is explored. The queries can contain unions, branching, label wildcards, the vertical child and descendant axes, the horizontal following and following-sibling axes, as well as positional (i.e., first and last) axes. Equivalence characterizations are provided, and their complexity is analyzed. As the descendant axis involves a recursive relationship, this article is also the first to address bag equivalence over recursive queries, in any setting.

1 Introduction

XPath [4] is a simple language for navigating XML documents. As such, it is an important component of many XML standards, including XSL [1], XQuery [2], XML Schema [19], XLink [11] and XPointer [10]. Proper understanding of the fundamentals of XPath (i.e., issues such as expressivity, optimization, equivalence) are a key to effective use of all the technologies above.

In this article we focus on the problem of determining equivalence of tree pattern queries, under bag semantics. Formally, given two tree pattern queries, the equivalence problem is to determine whether the queries will yield the same results, over any database. Since tree patterns can be used as an abstract model for XPath, our results yield equivalence characterizations for XPath queries.

Containment and equivalence for various fragments of XPath have been studied extensively, e.g., [18, 16, 15, 21, 12], as these problems are considered a key to query optimization and view usability. Containment of fragments of XPath, with respect to integrity constraints was studied in [12] and containment in the presence of DTDs was studied in [21]. Containment of XPath queries including branching, wildcard labels and the descendant axis was shown to be Co-NP complete in [15]. Containment in the presence of DTDs, disjunctions and variables (comparisons) was studied in [16]. Finally, containment of XPath 2.0 queries (which can include path intersection, path equality, path complementation, for-loops and transitive closure) was studied in [18]. All the above-mentioned work only considers queries evaluated under set semantics.
The XPath standard dictates that XPath is evaluated under *set semantics*. Intuitively, this means that a node will be in the result at most one time, regardless of the number of ways that the XPath query can be satisfied while deriving this node. SQL, on the other hand, provides the user with flexibility (by choosing to include or omit the DISTINCT keyword) in deciding whether queries should be evaluated under set semantics, or under *bag semantics*, wherein answers are returned as many times as they have derivations. Such flexibility is useful, especially when aggregation functions are applied to the data. In fact, [3] went so far as to refer to queries evaluated under bag semantics as *real queries*. The following example demonstrates information needs that are properly captured by using bag semantics.

**Example 1.1** The database in Figure 1 describes the structure of departments within a (software) company. A department may have several teams, each of which has a leader and direct members. A team may further be composed of sub-teams, again with their own leader and members, and so on. Thus, in the example database, Sally leads a team with direct member Jim. Indirect members of this team include Saul (who is himself a team leader), as well as John, Jake and Jessy. The nodes are numbered for easy reference.

Figure 2 contains several queries over this database. We follow the standard convention of depicting queries as tree patterns with single and double lines representing the child axis and descendant axis, respectively. The ovals indicate output nodes.

When evaluated under bag semantics, the bag union of queries $Q_1$ and $Q_2$, 

![Diagram](image)
will return each department, as many times as it has leaders and members. This is useful, as the result can be used to count the number of employees per department. In other words, the number of times that a department is returned is precisely the count value for the number of employees in the department. Query $Q_3$ returns each leader as many times as the number of (direct and indirect non-leader) members within his team, which can be used to count the team size under the responsibility of each leader. Finally, $Q_4$ returns each member, as many times as the number of teams in which the member (directly or indirectly) belongs. Again, this can be used to count the size of the chain of command, above each member.

Equivalence of Datalog queries under bag semantics has been studied rather extensively [3, 13, 7, 6, 5] (sometimes in the context of count-queries). However, these results do not carry over to tree pattern queries, for several reasons. First, tree pattern queries are evaluated over databases that must have tree form, whereas Datalog queries are evaluated over arbitrary databases. Second, tree patterns use axes, such as descendant, that are inherently recursive. No previous work has studied bag equivalence of recursive Datalog queries. Third, bag and bag-set semantics coincide for tree pattern queries (as each node has a unique identity in the XML data model), but not for Datalog queries—making the setting quite different. This article is the first to study equivalence of tree pattern queries (and thereby, of XPath) under bag semantics. We note that due to the well-known correspondence between bag semantics and queries with the aggregation function count [6], the results in this article also imply equivalence characterizations for some forms of XPath counting queries.

The main contribution of this article is a complete characterization of bag equivalence of XPath queries, written as tree patterns. We consider queries that can contain multiple output nodes, unions, branching, label wildcards, the vertical child and descendant axes, the horizontal following, and following-sibling axes, as well as positional axes (first and last). The complexity of equivalence is also analyzed. As the descendant axis involves a recursive relationship, this article is the first to address bag equivalence over recursive queries, in any set-

---

$Q_1$ Company

Dept

Leader

$Q_2$ Company

Dept

Member

$Q_3$ Company

Team

Leader

Member

$Q_4$ Company

Dept

Member

Figure 2: Example queries. Ovals indicate output nodes.
An additional contribution of this article is an undecidability result for containment of XPath queries, evaluated under bag semantics, even for queries with no horizontal axes.

A preliminary version of these results appears in [8]. This article extends [8] in two significant ways. First, all proofs were omitted from [8], and appear fully here. Second, this article considers an alternative method of characterizing equivalence via a reduction to path equivalence of nondeterministic finite automata. This proof technique is interesting in itself, and also gives rise to a more efficient method of checking for equivalence. Thus, Section 5 contains all new material.

This article is structured as follows. Section 2 defines the notion of a database, a query and bag semantics for evaluation. Section 3 reduces the equivalence problem to that of equivalence of completely ordered sets of queries, and Section 4 provides our equivalence characterization. Section 5 provides an alternative equivalence characterization based on nondeterministic finite automata. Section 6 extends our query language (and results) to additional horizontal axes. Finally, Section 7 concludes.

2 Definitions

2.1 Databases

Let $\Sigma$ be an infinite set of symbols $A, B, C, \ldots$, called labels. A database $D = (V, E, r, \prec, \lambda)$ is a labeled, ordered, directed, rooted tree, where (1) $V$ is the set of nodes, (2) $E$ is the set of edges, (3) $r \in V$ is the root node, (4) $\prec$ is a complete ordering over sibling nodes and (5) $\lambda : V \rightarrow \Sigma$ associates each node $a$ with a label $\lambda(a)$. For sibling nodes $a, b$, we write $a \prec b$ if $a$ precedes $b$.

We say that $a$ is an ancestor of $b$ if there is a directed path from $a$ to $b$ in $D$. We say that $a$ is a non-strict ancestor of $b$ if $a$ is an ancestor of $b$ or $a = b$. We will also say that $b$ c-follows $a$ if a depth-first traversal of $D$ starting at $c$ reaches $a$, returns to $c$, and then reaches $b$. Formally, $b$ c-follows $a$ if there are children $a'$ and $b'$ of $c$ such that (1) $a'$ is a non-strict ancestor of $a$, (2) $b'$ is a non-strict ancestor of $b$ and (3) $a' \prec b'$.

Example 2.1 Consider the database in Figure 1. Assume that the ordering $\prec$ is defined in the left-to-right order in which nodes appear. We use $a_i$ to denote the node labeled $i$. Then, $a_7$ is an ancestor of $a_9$ and a non-strict ancestor of $a_7$ and $a_9$. Node $a_{12}$ $a_7$-follows $a_8$, as does $a_{10}$. However, $a_{12}$ does not $a_7$-follow $a_{11}$.

2.2 Query Syntax

We formally define a tree pattern query, called a query for short, as follows.\(^2\)

\(^2\)The query language is enriched in Section 6.
Definition 2.2 (Query) A query $Q = (V, E, r, \prec, \lambda, \bar{o})$ is defined similarly to a database, with four adaptations:

- the set of edges $E$ is a disjoint union of two sets $E_1$ and $E_2$, called child and descendant edges, respectively;
- $\lambda$ is a function $V \rightarrow \Sigma \cup \{\ast\}$, where $\ast$ is a special symbol not appearing in $\Sigma$, called the wildcard symbol;
- $\prec$ is a partial ordering among sibling nodes in $V$, i.e., there may be sibling nodes $v, w$ for which neither $v \prec w$ nor $w \prec v$;
- $\bar{o}$ is a sequence of nodes from $V$, called the output nodes.

To make the presentation clear, we use lowercase letters from the beginning of the alphabet $a, b, c, \ldots$ to denote database nodes, lowercase letters from the end of the alphabet $u, v, w, \ldots$ to denote query nodes and capital letters from the beginning of the alphabet $A, B, C, \ldots$ to denote symbols from $\Sigma$.

We differentiate two special types of queries. If $\prec$ is a complete ordering of sibling nodes, we say that $Q$ is a completely ordered query. If $\prec$ is empty, then $Q$ is an orderless query. We use $Q$ to denote a multiset of queries. If all the queries in $Q$ are completely ordered (resp. orderless), then we say that $Q$ is completely ordered (resp. orderless).

2.3 Query Semantics

In this article, we evaluate queries over databases under bag semantics. In other words, a tuple of nodes may appear multiple times in the output, depending on the number of matchings achieving this tuple. A formal definition follows.

Definition 2.3 (Matching) Let $D = (V_D, E_D, r_D, \prec_D, \lambda_D)$ be a database and $Q = (V_Q, E_1 \cup E_2, r_Q, \prec_Q, \lambda_Q, \bar{o})$ be a query. A mapping $\mu : V_Q \rightarrow V_D$ is a matching of $Q$ to $D$ if all the following conditions hold:

- edge consistency: $\forall (v, w) \in E_1$, $(\mu v, \mu w) \in E_D$ and $\forall (v, w) \in E_2$, the node $\mu w$ is an ancestor of $\mu v$ in $D$;
- root consistency: $\mu r_Q = r_D$;
- label consistency: $\forall v \in V_Q$, either $\lambda_Q(v) = \ast$ or $\lambda_Q(v) = \lambda_D(\mu v)$;
- order consistency: $\forall v, w \in V_Q$, if $v, w$ are sibling nodes with parent $u$ and $v \prec_Q w$, then $\mu w$ $\mu u$-follows $\mu v$, i.e., there exist children $a$ and $b$ of $\mu u$ in $V_D$ such that
  1. $a$ is a non-strict ancestor of $\mu v$,
  2. $b$ is a non-strict ancestor of $\mu w$ and
  3. $a \prec_D b$.
Note that if $Q$ is completely ordered, then $\mu$ must be injective. We use $\mathcal{M}(Q, D)$ to denote the set of all matchings from $Q$ to $D$.

The result of applying $Q$ to $D$, is the multiset of tuples

$$Q(D) := \left\{ \mu(\bar{o}) \mid \mu \in \mathcal{M}(Q, D) \right\},$$

where $\{\cdot\}$ is used to denote a multiset. The result of applying a multiset of queries $Q$ to $D$, is simply the bag-union of the results of applying all queries $Q \in Q$ to $D$, i.e.,

$$Q(D) := \biguplus_{Q \in Q} Q(D).$$

**Example 2.4** Consider the database $D$ in Figure 1. There are three matchings of $Q_1$ to $D$, all of which map Company to $a_1$, and Dept to $a_2$. These matchings map Leader to $a_5$, $a_8$ and $a_{11}$. Similarly, there are five matchings of $Q_2$ to $D$, all of which agree with $Q_1$ on Company and Dept, but map Member to one of five different nodes. Thus, $\{Q_1, Q_2\}(D)$ contains $a_2$ a total of 8 times, i.e., can be used to count the number of employees per department.

Now consider $Q_3$. For each Leader node in $D$, there is a single way to map Team, and multiple ways to map Member (exactly as many as there are Member descendant nodes for the Team). Thus, $Q_3(D)$ contains $a_5$, $a_8$ and $a_{11}$ a total of 5, 4 and 3 times, respectively. It is then easy to see that $Q_3$ returns each leader exactly as many times as the number of (direct or indirect non-leader) members of his team.

Finally, consider $Q_4$. For each Member node in $D$, there is a matching of $Q_4$ to the database that maps the node labeled $\ast$ to each of the ancestors of the member node (other than Company and Dept). Therefore, each Member node will be returned exactly as many times as the number of teams to which the member (directly or indirectly) belongs.

**Remark 2.5** Our semantics for $\prec$ differs slightly from the standard semantics of the XPath following axis. In particular, given query node $u$ with children $v$ and $w$ such that $v \prec w$, we require $\mu v$ and $\mu w$ be non-strict descendants of distinct children $a$ and $b$ of $\mu u$. The standard meaning of the following axis would only require $\mu v$ and $\mu w$ to have a common ancestor that is a descendant of $\mu _u$, and differs from $\mu v$.

Our semantics for $\prec$ allows the equivalence characterization to be presented in a clear fashion. It is easy to express the semantics of the XPath following axis (as well as following-sibling) using the $\prec$ primitive. Details appear in Section 6.

**2.4 Problems of Interest**

We say that the multisets of queries $Q$ and $Q'$ are equivalent, written $Q \equiv Q'$, if, for all databases $D$, it holds that $Q(D) = Q'(D)$. Similarly, $Q$ is contained in $Q'$, written $Q \subseteq Q'$, if, for all databases $D$, it holds that $Q(D)$ is a subbag of $Q'(D)$. The bag-equivalence problem (or simply equivalence problem, for short) is
to determine whether two multisets of queries are equivalent. The containment problem is defined similarly.

In this article we focus on the equivalence problem, and provide a complete characterization of equivalence. Limited results are also presented on the containment problem.

3 Reduction to Completely Ordered Queries

In this section we reduce the general problem of equivalence of multisets of queries to that of equivalence of multisets of completely ordered queries. Formally, we show that for any query \( Q \) there exists a multiset of completely-ordered queries \( \overline{Q} \) that is equivalent to \( Q \). Note that care must be taken to ensure that \( \overline{Q} \) not only returns the same results as \( Q \), but also with the same multiplicities.

The key concept used in finding \( \overline{Q} \) is that of an expansion, defined next.

Let \( Q = (V, E, r, \prec, \lambda, \overline{o}) \) be a query. We say that a pair of nodes \( v, w \) are unordered sibling nodes if (1) \( v \) and \( w \) are siblings and (2) neither \( v \prec w \) nor \( w \prec v \) follows from \( \prec \). If \( v, w \) are unordered sibling nodes, then the \( \{v, w\} \)-expansion of \( Q \) is the multiset of queries \( \text{Exp}_{\{v,w\}}(Q) \) containing the following queries:

- **Directly Adding Order:** \( \text{Exp}_{\{v,w\}}(Q) \) contains the queries \( Q_1 \) and \( Q_2 \), which are derived by adding \( v \preceq w \) and \( w \preceq v \), respectively, to \( Q \);

- **Merging Nodes:** If \( \lambda(v) = \lambda(w) \), \( \lambda(v) = * \) or \( \lambda(w) = * \), then \( \text{Exp}_{\{v,w\}}(Q) \) contains the query \( Q_3 \), which is derived by
  1. removing \( v \) and \( w \) from \( V \), and adding the node \( \{v, w\} \), instead;
  2. giving \( \{v, w\} \) the label \( \lambda(v) \), if \( \lambda(v) \neq * \), and \( \lambda(w) \) otherwise;
  3. replacing every occurrence of \( v \) and \( w \) in \( E, E_j, \prec, \overline{o} \) with \( \{v, w\} \);  
  4. if \( (u, \{v, w\}) \) is now in both \( E \) and \( E_j \), then removing \( (u, \{v, w\}) \) from \( E \).

In other words, nodes \( v, w \) are **merged** in \( Q_3 \).\(^3\)

- **Lowering a Single Node:** If \( (u, v) \in E \), then \( \text{Exp}_{\{v,w\}}(Q) \) contains the query \( Q_4 \), which is derived by (1) removing \( (u, v) \) from \( E \), (2) adding \( (w, v) \) to \( E \), and (3) replacing all occurrences of \( v \) in \( \prec \) with \( w \). In other words, in \( Q_4 \), the node \( v \) is **lowered** to become a child of \( w \).

  Similarly, if \( (u, w) \in E \), then \( \text{Exp}_{\{v,w\}}(Q) \) contains the query \( Q_5 \), which is created symmetrically, by lowering \( w \) to become a child of \( v \).

- **Lowering Both Nodes:** If \( (u, v) \in E \) and \( (u, w) \in E \), then \( \text{Exp}_{\{v,w\}}(Q) \) contains the query \( Q_6 \), which is derived by

\(^3\)Due to the symmetrical nature of \( Q_3 \), we do not need to create an additional query with the roles of \( v \) and \( w \) reversed.
Figure 3: Query $Q$ and $\{v, w\}$-expansion $\text{Exp}_{\{v, w\}}(Q) = \{Q_1, \ldots, Q_7\}$.

1. creating a new node $y$ with $\lambda(y) = *$;
2. replacing all occurrences of $v$ and $w$ in $\prec$ with $y$
3. adding edges $(u, y), (y, v)$ and $(y, w)$ to $E_f$;
4. removing edges $(u, v)$ and $(u, w)$ from $E_f$;
5. adding $v \prec w$ to $\prec$.

In other words, in $Q_6$, the nodes $v, w$ are lowered below a new node $y$.

Similarly, $\text{Exp}_{\{v, w\}}(Q)$ contains the query $Q_7$, created in the same fashion as $Q_6$, except in the final step $w \prec v$ is added to $\prec$.

Example 3.1 To demonstrate, consider the query $Q$ in Figure 3. The multiset $\text{Exp}_{\{v, w\}}(Q)$ contains exactly the queries $Q_1, \ldots, Q_7$. Note that if $Q$ were of a different form, then $\text{Exp}_{\{v, w\}}(Q)$ might contain less queries. For example, if the incoming edge of $v$ was not a descendant edge, then $\text{Exp}_{\{v, w\}}(Q)$ would not contain $Q_4, Q_6$ or $Q_7$.

Remark 3.2 In Figure 3, as in many of the upcoming figures, we do not explicitly note the output nodes. The reader may assume that $Q$ (and hence, $Q_1, \ldots, Q_7$) is Boolean, i.e., returns the empty sequence over a database $D$ with the multiplicity of the number of matchings of $Q$ over $D$. We choose to present Boolean queries in our examples, to reduce clutter.
There may be queries in $Exp_{(v,w)}(Q)$ that are isomorphic.

**Definition 3.3 (Isomorphic)** Queries $Q_1 = (V_1, E_{(v,w)} \cup E_{1}, r_1, \prec_1, \lambda_1, o_1)$ and $Q_2 = (V_2, E_{(v,w)} \cup E_{2}, r_2, \prec_2, \lambda_2, o_2)$ are isomorphic, denoted $Q_1 \sim Q_2$, if there exists a bijective mapping $\varphi$ from $V_1$ to $V_2$ such that

- $(v, w) \in E_{(v,w)}$ if and only if $(\varphi(v), \varphi(w)) \in E_{(v,w)}$;
- $(v, w) \in E_{1}$ if and only if $(\varphi(v), \varphi(w)) \in E_{2}$;
- $\varphi r_1 = r_2$;
- $v \prec_1 w$ holds if and only if $\varphi v \prec_2 \varphi w$ holds;
- for all $v$, $\lambda_1(v) = \lambda_2(\varphi(v))$;
- $\varphi o_1 = o_2$.

For example, in Figure 3, there are three pairs of isomorphic queries, namely, $Q_1 \sim Q_2$, $Q_4 \sim Q_5$, and $Q_6 \sim Q_7$.

We now show the main property of a $\{v, w\}$-expansion.

**Lemma 3.4** Let $Q$ be a query and $v, w$ be unordered sibling nodes. Then, $Q \equiv Exp_{(v,w)}(Q)$.

**Proof.** Let $D$ be a database. Let $M_1$ and $M_2$ be the sets of matchings of $Q$ and $Exp_{(v,w)}(Q)$, respectively, to $D$. Observe that queries in $Exp_{(v,w)}(Q)$ may differ from $Q$ (and from one another) in their tuples of output. This occurs when nodes are merged (i.e., $Q_3$). However, the output tuples of nodes in all queries in $\{Q\} \cup Exp_{(v,w)}(Q)$ are of the same length. Given a mapping $\mu \in M_1 \cup M_2$ that maps a query $Q'$ to $D$, we will use $\mu_0$ to denote the projection of $\mu$ on the output tuple of $Q'$.

We show that there is a bijective mapping $\pi$ from $M_1$ to $M_2$ such that for all $\mu \in M_1$, we have that $\mu_0 = (\pi \mu)_0$. This is sufficient to prove our claim. Let $\mu$ be a mapping in $M_1$. We consider several cases depending on how $\mu$ maps $v, w$ and their parent $u$.

- If $\mu(u)$ and $\mu(w)$-follows $\mu(v)$ in $D$, then $\mu$ also satisfies $Q_1$, (and $Q_1$ has the same output tuple as $Q$), and we define $\pi(\mu) = \mu$. Similarly, if $\mu(v)$ and $\mu(w)$-follows $\mu(u)$, then $\mu$ also satisfies $Q_2$, and $\pi(\mu) = \mu$.
- If $\mu(v) = \mu(w)$, then let $\mu'$ the mapping defined by removing the mappings of $v$ and $w$ from $\mu$, and adding the mapping $\mu'({\{v, w}\}) = \mu(v)$. We choose $\pi(\mu) = \mu'$ and obviously $\mu'$ satisfies $Q_3$, and $\mu_0 = \mu'_0$.
- If $\mu(v)$ is a descendant (resp. ancestor) of $\mu(w)$ in $D$, then $\pi(\mu) = \mu$ and $\mu$ satisfies $Q_4$ (resp. $Q_5$).
Algorithm ComputeCompleteExpansion(Q)

1: while ∃Q ∈ Q that is not completely ordered do
2:   Choose a pair of llu sibling nodes v, w
3:   Q ← Q \ {Q} ∪ Exp{v,w}(Q)
4: return Q

Figure 4: Algorithm for computing a complete expansion.

- If μ(w) a-follows μ(v) (resp. μ(v) a-follows μ(w)), for some descendant a of μ(u) in D, then let μ' be the mapping defined by extending μ with μ'(y) = a. We choose π(μ) = μ'. Observe that μ' satisfies Q₆ (resp. Q₇).

  It is important to note that there can only be a single node a such that μ(w) a-follows μ(v), since by definition, a must be the lowest common ancestor of μ(v) and μ(w).

Clearly, π is injective, as each Qi defines a different relative ordering among u, v and w. It is also surjective, since the queries in Exp{v,w}(Q) simply impose a relationship between μ(v) and μ(w) (beyond the constraints already appearing in Q), and only relationships consistent with Q are imposed.

A pair u, v of sibling nodes are at level n if u and v each have n ancestors. We say that u, v are a lowest level pair of unordered sibling nodes, or llu sibling nodes for short, if u, v are unordered sibling nodes and there is no pair of unordered sibling nodes at a lower level (i.e., closer to the root) than u, v. Note that there may be several lowest-level pairs of unordered sibling nodes. A complete expansion of a multiset of queries Q, denoted Exp(Q), is derived by the process described in Figure 4.

The following theorem states that the process of computing a complete expansion terminates (i.e., Exp(Q) will be finite) and, moreover, that each query in Exp(Q) is at most twice as large as Q. Of course, Exp(Q) can contain an exponential number of queries. Finally, note that termination is not obvious, as the process of computing a complete expansion introduces new node, which must be considered in later stages of computing a complete ordering.

Theorem 3.5 Let Q be a query. Then, a complete expansion Exp(Q) is of finite size. Moreover, each query in Exp(Q) is at most twice as large as Q.

Proof. Let v, w be an unordered sibling pair in Q. Suppose that v, w are at level n. Our proof of termination is based on the following property. For every query Q' in Exp{v,w}(Q),

\[ \text{Unordered}_n(Q') < \text{Unordered}_n(Q), \tag{1} \]
where \( \text{Unordered}_{\leq n}(Q) \) denotes the number of unordered sibling pairs at any level \( \leq n \).

Equation 1 is obvious for queries \( Q_1, Q_2, Q_3 \) of \( \text{Exp}_{\{v,w\}}(Q) \). For queries \( Q_4, Q_5, Q_6, Q_7 \), observe that the number of pairs of unordered sibling nodes at level \( \leq n \) has been reduced by at least one, as the pair \( v, w \) is no longer unordered and no new unordered pairs of sibling nodes at level \( \leq n \) have been introduced. Similarly, for queries \( Q_6, Q_7 \), the number of pairs of unordered sibling nodes at level \( \leq n \) has been reduced by at least one, as the pair \( v, w \) is no longer unordered. In addition, the new node \( y \) does not increase the number of pairs of unordered sibling nodes as, for any node \( z \) at level \( n \) such that \( z, y \) are unordered sibling nodes in \( Q_6 \) or \( Q_7 \) the pairs \( z, v \) and \( z, w \) were both unordered sibling nodes in \( Q \).

Now, consider the non-deterministic process of computing a complete expansion of \( Q \). We can view a specific execution of this process as creating a tree \( T \), whose nodes are queries, in the following fashion:

- The root of \( T \) is \( Q \).
- For any node \( Q' \) in \( T \), if we choose to replace \( Q' \) with \( \text{Exp}_{\{v,w\}}(Q') \) during the execution, then each query in \( \text{Exp}_{\{v,w\}}(Q') \) is a child of \( Q' \) in \( T \).

We will show that \( T \) is of finite depth, and hence, the process of computing a complete expansion of \( Q \) terminates. Specifically, we show that the depth of \( T \) is bounded by \( 8m^3 \), where \( m \) is the number of nodes in \( Q \).

We first observe that for all \( Q' \) in \( T \), the number of nodes in \( Q' \) is at most \( 2m \). To see this, it is easy to show (by induction on the depth of \( Q' \)) that every leaf node in \( Q' \) is either a node in \( Q \), or the result of merging several nodes of \( Q \). Therefore, the number of leaf nodes in \( Q' \) is at most \( m \). In addition, newly introduced nodes in the expansion process (i.e., \( y \) in \( Q_6, Q_7 \)) will always have at least two children. This follows since when \( y \) is introduced it has two children with an ordering already determined between these nodes. From the above observations it immediately follows that the number of newly introduced nodes in \( Q' \) is bounded by \( m \), i.e., \( Q' \) has at most \( 2m \) nodes.

Let \( Q' \) be a query in \( T \). Recall that a lowest-level pair of unordered sibling nodes is always chosen during the expansion process, i.e., to create the children of \( Q' \) in \( T \). Let the level of this pair be \( n \). Since every query in \( T \) has at most \( 2m \) nodes, in particular \( Q' \) has at most \( 2m \) nodes at level \( n \), and thus, at most \( 4m^2 \) pairs of unordered sibling nodes at level \( n \). By Equation 1, every query \( Q'' \) of distance greater or equal to \( 4m^2 \) from \( Q' \) will have no unordered sibling pairs at level less than or equal to \( n \).

Since all queries in \( T \) have at most \( 2m \) nodes, they are at most \( 2m \) in height. Using the argument above it follows that the height of \( T \) (which is determined by the number of times that a pair is chosen for expansion) is at most \( 2m(4m^2) = 8m^3 \).

We now conclude the main result of this section.

\[ \square \]
Figure 5: Query $Q$ and complete expansion $\text{Exp}(Q) = \{Q_1, \ldots, Q_8\}$.

**Corollary 3.6** Let $Q$ be a multiset of queries. Then, $\text{Exp}(Q)$ is a finite completely ordered multiset of queries, such that $Q \equiv \text{Exp}(Q)$. Moreover, each query in $\text{Exp}(Q)$ contains at most twice as many nodes as the largest query in $Q$.

**Example 3.7** Consider first query $Q$ in Figure 3. All queries in $\{Q_1, \ldots, Q_7\}$ are completely ordered, and thus, $\text{Exp}(Q) = \{Q_1, \ldots, Q_7\}$. Note that a complete expansion was derived by a single step of the expanding process.

A more sophisticated example appears in Figure 5. Consider the query $Q$ in this figure. It is possible to show that $\text{Exp}(Q) = \{Q_1, \ldots, Q_8\}$. Note that these queries are derived by repeatedly choosing pairs of unordered sibling nodes, and computing the expansion for these pairs. (The numbering of the queries $Q_1$ through $Q_8$ is only provided for convenience and does not correspond with the numbering provided in the definition of a $\{v, w\}$-expansion.) By Corollary 3.6, it follows that $Q \equiv \{Q_1, \ldots, Q_8\}$. 

12
Characterizing Equivalence

In this section, we study equivalence of multisets of completely ordered queries. Therefore, unless explicitly stated otherwise, we will assume that all queries are completely ordered. By Corollary 3.6, equivalence of general multisets of queries can be reduced to equivalence of multisets of completely ordered queries.

We start by introducing the notion of a core of a query, which captures the essence of a query (Section 4.1). Next, we consider several cases in which multisets of queries appear different, yet are equivalent (Section 4.2). Then, we introduce the notion of a canonical database, which will be used in the proof of our equivalence characterization (Section 4.3). Finally, we present and prove a sufficient and necessary property for equivalence (Section 4.4).

4.1 Core Nodes and Queries

In this section we define cores of queries. Intuitively, a core of a query $Q$ captures the “essence” of $Q$ while abstracting away (removing) paths of wildcard labeled nodes. To formalize this idea, we start by defining core nodes.

**Definition 4.1 (Core Node)** Let $Q = (V, E, r, \prec, \lambda, \bar{o})$ be a completely ordered query. We say that a node $v \in V$ is a core node if at least one of the following conditions hold:

1. $v$ is the root, i.e., $v = r$;
2. $\lambda(v) \neq \ast$;
3. $v$ has more than one child;
4. $v$ is an output node, i.e., appears in $\bar{o}$;
5. $v$ is a leaf node.

We use $\hat{V}$ to denote the set of core nodes in $V$.

Consider a path in $Q$ of the form $p = u, z_1, \ldots, z_n, v$ where $u, v \in \hat{V}$, and $z_i \not\in \hat{V}$, for all $i$. We call $p$ a core path, $u, v$ core endpoints (as they are at either ends of a core path) and $z_1, \ldots, z_n$ intermediate nodes. In particular, in a core path all intermediate nodes $z_i$ have a single child and are labeled with $\ast$. The notion of a core path will be central to many of the results in this section.

We use $Q_{\hat{u}(u,v)}$ to denote the query derived by collapsing the path from $u$ to $v$, i.e., $Q_{\hat{u}(u,v)}$ is derived from $Q$ by:

- removing all nodes $z_1, \ldots, z_n$;
- adding a descendant edge $(u, v)$;\(^4\)
- replacing any occurrence of $z_1$ in $\prec$ by $v$ (i.e., $v$ inherits all sibling orderings of $z_1$).

\(^4\)For the special case that $n = 0$, i.e., there are no intermediate nodes, and $(u, v)$ is a child edge, it is replaced with a descendant edge.
Figure 6: Query $Q$, and its core $\hat{Q}$. The core of $Q'$ is isomorphic to $Q$. In $Q$ and $Q'$, core nodes are underlined.

Obviously, the query $Q_{(u,v)}$ is completely ordered, if $Q$ is completely ordered. The core query of $Q$, denoted $\hat{Q}$ is the query derived by collapsing all core paths of $Q$.

To demonstrate the notion of a core query, consider the query $Q$ and its core $\hat{Q}$ in Figure 6. The core nodes are underlined. Note that the node $y$ does not appear in $\hat{Q}$, as $y$ is not a core node. Note also that $x \prec v$ in $\hat{Q}$, as $x \prec y$ in $Q$.

Different queries may have isomorphic cores. Formally, we say that $Q$ and $Q'$ are core isomorphic, if $\hat{Q}$ is isomorphic to $\hat{Q}'$. Suppose that $Q$ and $Q'$ are completely-ordered, core-isomorphic queries. The queries $\hat{Q}$ and $\hat{Q}'$ are also completely ordered. Therefore, there is a single isomorphism $\varphi$ from $\hat{Q}$ to $\hat{Q}'$.

By abuse of notation, we will consistently use the same letters to denote a core node $v$ in $Q$ and its (single) corresponding node $\varphi v$ in $Q'$. Thus, e.g., if $u, \ldots, v$ is a core path in $Q$, then $u, \ldots, v$ will be a core path in $Q'$ with corresponding endpoints. (Note, of course, that the intermediate nodes in the core path will differ.)

Now, consider query $Q'$ in Figure 6. Once again, the core nodes have been underlined. Observe that we have used the same letters to denote corresponding core nodes in $Q'$ and in $\hat{Q}$, as $Q \sim Q'$. Using this notational convention, $\hat{Q}$ and $Q'$ are actually identical (and are exactly the query in the center of Figure 6).

4.2 Different, Yet Equivalent, Queries

We extend the notion of isomorphic queries, presented earlier, to multisets of queries. Formally, we say that two multisets of queries $Q$ and $Q'$ are isomorphic, written $Q \sim Q'$ if (1) $|Q| = |Q'|$ and (2) there is a bijection $\pi$ from $Q$ to $Q'$ such that

$$\forall Q \in Q(Q \sim \pi(Q)).$$
Obviously, if \( Q \sim Q' \), then \( Q \equiv Q' \). Unfortunately, the converse does not hold. In this part we explore two simple cases where multisets of non-isomorphic queries are indeed equivalent. Later on we will show that, essentially, these are the only cases in which non-isomorphic queries can be equivalent.

### 4.2.1 Flipping Edges Types

Let \( Q \) be a query. Let \( u, z_1, \ldots, z_n, v \) be a core path in \( Q \). We say that a node \( z_i \) on this path is flippable if the single incoming and outgoing edges of \( z_i \) are of different types (i.e., one is a child edge and one is a descendant edge). We say that \( Q' \) is the \( z_i \)-flip of \( Q \), if \( Q' \) is derived from \( Q \) by switching the types of the incoming and outgoing edges of \( z_i \) (e.g., if \( (z_{i-1}, z_i) \in E_j \) and \( (z_i, z_{i+1}) \in E_j \) in \( Q \), then \( (z_{i-1}, z_i) \in E_j' \) and \( (z_i, z_{i+1}) \in E_j' \) in \( Q' \)).

To demonstrate, consider queries \( Q \) and \( Q' \) from Figure 7. It is easy to see that \( Q' \) is the \( y \)-flip of \( Q \). Next we will show that this implies that \( Q \equiv Q' \).

**Proposition 4.2** Let \( Q \) be a query, and \( z \) be a flippable intermediate node on a core path. Let \( Q' \) be the \( z \)-flip of \( Q \). Then, \( Q \sim Q' \) and \( Q \equiv Q' \).

**Proof.** Clearly \( \hat{Q} \) and \( \hat{Q'} \) are identical, and hence, \( \hat{Q} \sim \hat{Q'} \). It remains to show that \( Q \equiv Q' \). Let \( D \) be a database, and \( \tilde{a} \) be a tuple of nodes the length of the output nodes in \( Q \) and \( Q' \). Let \( M \) (resp. \( M' \)) be the set of matchings of \( Q \) (resp. \( Q' \)) to \( D \) that map the output nodes to \( \tilde{a} \). We must show that \( |M| = |M'| \).

Let \( u \) be the single parent of \( z \) and \( v \) be the single child of \( z \). (Note that \( z \) has a single child since \( z \) is not a core node.) Suppose, without loss of generality, that \( (u, z) \) is a child edge in \( Q \) and \( (z, v) \) is a descendant edge in \( Q \). Let \( \mu \) be a mapping in \( M \). We define the mapping \( \mu' \) from \( Q' \) to \( D \) as follows:

\[
\mu'(w) = \begin{cases} 
\mu(w) & w \neq z \\
p(\mu(v)) & w = z \text{ and } p(\mu(v)) \text{ is the parent of } \mu(v) \text{ in } D
\end{cases}
\]

It is easy to see that \( \mu' \) must be in \( M' \). (Note that the output nodes in \( \mu' \) are mapped to \( \tilde{a} \) since \( z \) cannot be an output node.)

Let \( \varphi \) be the mapping that associates each matching \( \mu \in M \) with a matching \( \mu' \in M' \) in the manner described above. We show that \( \varphi \) is injective. Consider mappings \( \mu_1, \mu_2 \in M \). Obviously, if \( \mu_1 \) and \( \mu_2 \) differ on any variable other than \( z \), then \( \varphi(\mu_1) \neq \varphi(\mu_2) \). Now, suppose that \( \mu_1 \) and \( \mu_2 \) differ on \( z \). Observe that in this case, \( \mu_1 \) and \( \mu_2 \) must also differ on \( v \); since \( \mu_1(z) \) and \( \mu_2(z) \) are different children of \( \mu(u) \), and hence, have disjoint sets of descendants. It follows that \( \varphi \) is an injective mapping, i.e., \( |M| \leq |M'| \).

Now, let \( \mu' \) be a mapping in \( M' \). We define the mapping \( \mu \) from \( Q \) to \( D \) as follows:

\[
\mu(w) = \begin{cases} 
\mu'(w) & w \neq z \\
c(\mu'(u)) & w = z \text{ and } c(\mu'(u)) \text{ is the child of } \mu'(u) \text{ that is an ancestor of } \\
\mu'(v) & \text{ in } D
\end{cases}
\]
Once again, it is easy to see that $\mu$ must be in $\mathcal{M}$. Let $\varphi'$ be the mapping that associates each matching $\mu' \in \mathcal{M}'$ with a matching $\mu \in \mathcal{M}$ in the manner described above. Using similar reasoning to the previous argument, one can show that $\varphi'$ is an injective mapping from $\mathcal{M}'$ to $\mathcal{M}$, i.e., $|\mathcal{M}'| \leq |\mathcal{M}|$.

It now follows that $|\mathcal{M}| = |\mathcal{M}'|$, as required. □

We say that queries $Q$ and $Q'$ are flip isomorphic, denoted $Q \sim_f Q'$, if

- $\hat{Q}$ is isomorphic to $\hat{Q}'$;
- for all $(u, v)$ in $\hat{Q}$, the path from $u$ to $v$ in $Q$ has the same number of child edges and the same number of descendant edges, as the path from $u$ to $v$ in $Q'$.\(^5\)

To understand the intuition behind this notion, it is easy to see that if $Q \sim_f Q'$, then there is a series of flips that, starting from $Q$, derives a query that is isomorphic to $Q'$. We extend the notion of flip isomorphic to multisets of queries in the natural way, i.e., $Q \sim_f Q'$ if there is a bijection $\pi$ from $Q$ to $Q'$ such that for all $Q \in Q$, we have $Q \sim_f \pi(Q)$.

Corollary 4.3 follows from Proposition 4.2.

**Corollary 4.3** Let $Q$ and $Q'$ be multisets of queries. If $Q \sim_f Q'$, then $Q \equiv Q'$.

### 4.2.2 Edge Unrolling

We now consider a second case where non-isomorphic queries can be equivalent. Let $Q = (V, E_f \cup E_g, r, \preceq, \lambda, \bar{o})$ be a query. Let $(u, v)$ be an edge in $E_g$. The $(u, v)$-unrolling of $Q$ is the set of queries $\{Q_1, Q_2\}$ derived as follows:

- $Q_1$ is simply the query $Q$, with the edge $(u, v)$ removed from $E_g$ and added to $E_f$.

\(^5\)Note that we are following the convention stated earlier that uses the same node names to denote corresponding nodes in queries with isomorphic cores.
• $Q_2$ is the query derived by (1) adding a node $z$ with $\lambda(z) = \ast$, (2) replacing all occurrences of $v$ in $\prec$ with $z$ and (3) removing edge $(u, v)$ from $E_j$ and adding edges $(u, z)$ to $E_j$ and $(z, v)$ to $E_j$. In other words, the node $v$ is demoted to be below the new node $z$.

Note that if $Q$ is completely ordered, then so is $\{Q_1, Q_2\}$.

**Example 4.4** To demonstrate, observe that the set $\{Q_1, Q_2\}$ in Figure 8 is the $(u, v)$-unrolling of $Q$ (in the same Figure). This figure demonstrates a single unrolling of an edge in $Q$. Obviously, it is possible to continue unrolling edges in $Q_1$ and $Q_2$, and creating additional queries.

Intuitively, $Q_1$ captures matchings $\mu$ for which $\mu(v)$ is a child of $\mu(u)$ and $Q_2$ captures matchings $\mu'$ for which $\mu'(v)$ is a descendant, but not a child, of $\mu'(u)$. Therefore, the following property is immediate.

**Proposition 4.5** Let $Q$ be a query, and $(u, v)$ be a descendant edge in $Q$. Then, the $(u, v)$-unrolling of $Q$ is equivalent to $Q$.

Let $Q$ be a query and $k$ be a positive integer. We say that $Q$ is $k$-unrolled if, for each core path $p$ in $Q$, at least one of the following conditions holds

• $p$ contains only child-edges or

• $p$ contains at least $k$ edges.

Similarly, we say that a multiset of queries $Q$ is $k$-unrolled, if every query in $Q$ is $k$-unrolled. Given a multiset of queries $Q$, the definition of a $(u, v)$-unrolling of a query immediately provides us with a method to create a multiset $Unroll_k(Q)$ that is $k$-unrolled and is equivalent to $Q$. Actually, there may be many different ways to create a $k$-unrolled multiset of queries equivalent to $Q$, as there may be several descendant edges on the core path from $u$ to $v$ (which can be unrolled). To make $Unroll_k(Q)$ unambiguous, we will assume that we always unroll the topmost descendant edge on the path.
We will write $Q \sim_k Q'$ if the $k$-unrollings of $Q$ and $Q'$ are flip isomorphic, i.e., if $\text{Unroll}_k(Q) \sim^f \text{Unroll}_k(Q')$. Later, we will prove that given multisets of completely ordered queries $Q$ and $Q'$ it is possible to determine a value $k$ such that $Q \equiv Q'$ if and only if $Q \sim_k Q'$. To show this, we will need to consider databases of a specific form, called canonical databases, which are introduced in the next subsection.

**Example 4.6** In Figure 8, the query $Q$ has two core paths, $u \ldots v$ and $u \ldots x$. Observe that the set $\{Q_1, Q_2\}$ is a 2-unrolling of $Q$, but is not a 3-unrolling of $Q$ (e.g., since the paths from $u$ to $v$ and from $u$ to $x$ in $Q_2$ contain descendant edges, but do not contain 3 edges). Observe also that $Q \sim_k \{Q_1, Q_2\}$ for all $k \geq 2$. □

### 4.3 Canonical Databases

A canonical database for a query $Q$ is created out of its core $\hat{Q}$. Intuitively, a canonical database for $Q$ is generated from $\hat{Q}$ by replacing wildcards with a label, and replacing edges with chains of nodes. In other words, while $\hat{Q}$ is derived from $Q$ by collapsing core paths, a canonical database is created by “expanding” descendant edges of $\hat{Q}$.

Formally, let $Q$ be a completely ordered query. Creating a canonical database out of $Q$ involves three types of operations:

- **Core Path Replacement**: Let $u, \ldots, v$ be a core path in $Q$. Let $i$ be a non-negative integer. The $i$-length path replacement for $u, v$ is derived from $Q$ by replacing the path $u, \ldots, v$ with a path $u, z_1', \ldots, z_i', v$ of child edges where $z_j'$ are new nodes labeled $\ast$. In addition, if $z$ is the first child of $u$ on the core path before the replacement and $z'$ is the first child of $u$ on the core path after the replacement, then we replace every occurrence of $z$ in $\prec$ by $z'$.

  If $\theta$ is a function that maps each pair of core endpoints in $Q$ to a non-negative integer, then the $\theta$-path replacement of $Q$, denoted $Q_{\theta}$, is derived by simply applying the $\theta(u, v)$-length path replacement for each core path $u, \ldots, v$. Note that core paths may be replaced by paths that are either longer or shorter.

- **Wild Card Elimination**: Let $Z$ be an unused label. The wildcard eliminated version of $Q$, denoted $Q_{\ast \Rightarrow Z}$, is derived by replacing all $\ast$ labels with the label $Z$.

- **Descendant Edge Elimination**: The descendant-edge eliminated version of $Q$, denoted $Q_{/ / \Rightarrow}$, is derived by replacing all descendant edges with child edges.

---

6Since $\Sigma$ is infinite, we may assume that there is some label $Z$ not appearing in any queries.
Finally, let $\theta$ be a mapping of all pairs of core endpoints in $Q$ to nonnegative integers. The canonical database for $\theta$ and $Q$, denoted $D^Q_\theta$, is defined as

$$D^Q_\theta := ((Q_\theta)_{\vDash Z})_{\vDash /}.$$  

In other words, to derive a canonical database for $\theta$ and $Q$ we simply apply all three of the above defined steps, in order. When $Q$ is clear from the context we will drop the superscript and simply write $D_\theta$.

**Example 4.7** Consider the queries $Q$, its core $\hat{Q}$ and query $Q'$, appearing in Figure 6. Figure 9 contains two canonical databases for $Q$, namely $D_{\theta_1}$ and $D_{\theta_2}$. Due to our notational convention of using the same letters for corresponding core nodes in core-isomorphic queries, we can also view these databases as being canonical databases for $Q'$. Note that $Q$ is not satisfiable over $D_{\theta_1}$, but is satisfiable over $D_{\theta_2}$.

**Remark 4.8** Note that all wildcards are replaced with the same unused label $Z$. It is actually not necessary to replace all the wildcards with the same unused label. In fact, all the proofs in this section will remain correct even if we replace wildcards with several different unused labels. We will use this fact later on in Section 5 to simplify some of the proofs.

We continue with our abuse of notation, by using the same letters to denote core nodes and their corresponding canonical database nodes (as in Figure 9). Recall that the series of output nodes $\bar{o}$ in $Q$ are all core nodes, and hence, appear in the database. We will have a particular interest in the series of database nodes $\bar{o}$. To distinguish these as nodes from the database (as opposed to their identically named corresponding query nodes), we will call this series of nodes the **target series** for $D^Q_\theta$. 

---

**Figure 9:** Canonical databases.
We now establish which types of queries may return the target series when evaluated over $D_\theta^Q$. In the following proposition, we use $|V(Q)|$ to denote the number of nodes in $Q$ and $|V_{\neq *}(Q)|$ to denote the number of nodes in $Q$ that have a label that differs from $\ast$.

**Proposition 4.9** Let $D_\theta^Q$ be a canonical database for $\theta$ and $Q$. Let $Q'$ be a query with the same number of output nodes as $Q$. If $Q'(D_\theta^Q)$ contains the target series, then one of the following conditions must hold:

1. $|V(Q')| < |V(\hat{Q})|$;
2. $|V(Q')| = |V(\hat{Q})|$, but $|V_{\neq *}(Q')| < |V_{\neq *}(\hat{Q})|$;
3. $\hat{Q} \sim Q'$.

**Proof.** Let $\mu$ be a matching of $Q'$ to $D_\theta^Q$ that maps its output nodes to the target series. Observe that (by our notational convention) some of the nodes of $D_\theta^Q$ can actually be seen as nodes of $\hat{Q}$. If $\mu$ maps all core nodes of $Q'$ to nodes of $\hat{Q}$, then the required would easily follow. However, this property of $\mu$ does not necessarily hold.

Now, let us consider different the types of nodes $v$ in $\hat{Q}$:

1. $v$ is the root: In this case, $\mu(v)$ must be the root of $D_\theta^Q$, which is a core node in $Q$.
2. $\lambda(v) \neq \ast$: In this case $\mu(v)$ is a node with a label that differs from $Z$. Again, by the definition of $D_\theta^Q$, $\mu(v) \in \hat{Q}$.
3. $v$ has more than one child: In this case $\mu(v)$ has more than one child. Once again, by the definition of $D_\theta^Q$, $\mu(v) \in \hat{Q}$.
4. $v$ is an output node: $\mu$ maps $v$ to a target node, which is also an output node of $Q$, and hence, is in $\hat{Q}$.

5. $v$ is a leaf: Only in this case, is it possible for $\mu(v)$ to not be a node in $\hat{Q}$. If indeed $\mu(v) \notin \hat{Q}$, we correct $\mu$ by mapping $v$ to a leaf descendant of $\mu(v)$. Note that we now map $v$ to a leaf node, which, by the definition of $D_\theta^Q$, must be a node in $\hat{Q}$.

Let $\mu'$ be the mapping derived by the above corrections to $\mu$.

Observe that $\mu'$ is injective (since $Q'$ is completely ordered). Therefore, $|V(Q')| \leq |V(\hat{Q})|$. If, in addition, $|V(Q')| < |V(\hat{Q})|$, then the first condition of the proposition holds. Suppose otherwise, i.e., it holds that $|V(Q')| = |V(\hat{Q})|$. Since $\mu'$ must map nodes not labeled by $\ast$ in $Q'$ to nodes not labeled with $\ast$ in $Q$, it follows that $|V_{\neq *}(Q')| \leq |V_{\neq *}(\hat{Q})|$. If, in addition, $|V_{\neq *}(Q')| < |V_{\neq *}(\hat{Q})|$, then the second condition of the proposition holds. Suppose otherwise, i.e.,

\[
\begin{align*}
|V(\hat{Q})| &= |V(\hat{Q})| \\
|V_{\neq *}(\hat{Q})| &= |V_{\neq *}(\hat{Q})|.
\end{align*}
\]

20
We now show that $\hat{Q} \sim \hat{Q}'$. By the definition of a core of a query, $\hat{Q}$ and $\hat{Q}'$ do not contain any child edges (rather only descendant edges). It now follows that $\mu'$ is an isomorphism from $\hat{Q}'$ to $\hat{Q}$ since (1) $\mu'$ must map nodes to nodes with the same labels due to Equation 2 and (2) $\mu'$ satisfies all structural conditions. The latter follows since $\mu$ satisfied all structural conditions, and $\mu'$ preserves the same relative ordering and ancestor-descendant relationships.

We say that $\theta$ is a path respecting mapping of core paths in $Q$ to nonnegative integers, if for all core paths $p = u, \ldots, v$ in $Q$,

- if the core path $p$ (a) contains only child edges and (b) $v$ is not a leaf labeled $\ast$, then $\theta(p)$ is the number of intermediate nodes of $p$;
- otherwise $\theta(p)$ is greater-or-equal-to the number of intermediate nodes of $p$.

This is an important property, since if $\theta$ is not path respecting for $Q$, then obviously $Q(D^Q_\theta)$ will not contain the target series. We also define the notion of strictly path respecting, which is identical to path respecting, except that Condition (b) is removed.

We now establish the number of times that a query $Q$ will return the target series over a canonical database $D^Q_\theta$. In particular, we will show that this number is a polynomial in the values that $\theta$ assigns.

Let $Q$ be a query and let $\theta$ be a path respecting mapping. We analyze the number of times that $Q$ will return the target series. Let $u, v$ be core endpoints of $Q$. We use $d_{u,v}$ and $c_{u,v}$ to denote the number of descendant and child edges, respectively, on the path from $u$ to $v$. We define

$$b_{u,v} = \begin{cases} 1 & \text{if } d_{u,v} = 0 \\ \frac{\theta(u,v) - c_{u,v} + 1}{d_{u,v}} & \text{if } v \text{ is a leaf, } \lambda(v) = \ast \text{ and } v \text{ is not in } \partial \\ \frac{\theta(u,v) - c_{u,v}}{d_{u,v} - 1} & \text{otherwise} \end{cases}$$

**Proposition 4.10** Let $Q$ be a query and $\theta$ be a path-respecting mapping. Let $E$ be the set of all pairs of core endpoints in $Q$. Then, $Q$ returns the target series over $D^Q_\theta$ with multiplicity

$$\Phi(Q, \theta) = \prod_{(u,v) \in E} b_{u,v}. \quad (3)$$

**Proof.** Let $D$ be the database $D^Q_\theta$. Let $v$ be a node in $V(\hat{Q})$ we say that $v$ is a flexible core node if all the following conditions hold: $v$ is a leaf node, $v$ is labeled $\ast$, and $v$ is not an output node. Otherwise, we say that $v$ is a regular core node.

Let $M$ be the set of matchings of $Q$ to $D$ that return the target series. Let $V_\gamma$ be the set of regular core nodes in $Q$. Recall that by our naming convention, the nodes in $V_\gamma$ also appear in $D$. First, we show that if $v \in V_\gamma$, then $\mu(v) \in V_\gamma$. (A similar claim is shown in the proof of Proposition 4.9.)
Clearly, this holds if \( v \) is the root (since the root must be mapped to the root), and if \( \lambda(v) \neq * \) (since all nodes of \( D \) not in \( V_r \) are given an unused label). Similarly, if \( v \) has more than one child, then \( \mu(v) \) must be in \( V_r \), since only these nodes have more than one child in \( D \). Finally, if \( v \) is an output node, \( \mu(v) \) must be the corresponding output node of \( Q \), and thus be in \( V_r \). Therefore, we can conclude that \( \mu(v) \in V_r \).

Since \( \mu \) must satisfy the structural constraints of \( Q \), and \( Q \) is completely ordered, it immediately follows that \( \mu \) is the identity mapping on \( V_r \). Formally, for all \( \mu \in \mathcal{M} \), and for every regular core node \( v \) in \( Q \), it holds that \( \mu(v) = v \).

Now, suppose that \( u, v \) are core endpoints in \( Q \). The mappings in \( \mathcal{M} \) may differ on how they map the intermediate nodes on the path from \( u \) to \( v \), as well as on how they map \( v \), if \( v \) is a flexible core node. However, given pairs of core endpoints \( u, v \) and \( u', v' \), the mappings of the nodes on the path from \( u \) to \( v \) and those on the path from \( u' \) to \( v' \) can be chosen independently (even if \( u = u' \)). Formally, this means that if \( \mu, \mu' \in \mathcal{M} \) then the mapping \( \mu'' \) defined as

\[
\mu''(v) = \begin{cases} 
\mu(v) & \text{if } v \text{ is not on the path from } u' \text{ to } v' \\
\mu'(v) & \text{otherwise}
\end{cases}
\]

is also in \( \mathcal{M} \). Therefore, it is sufficient to analyze the number of ways that each core path can be mapped to nodes in the database. The product of these values will be exactly the number of matchings in \( \mathcal{M} \).

Consider first a core path \( u, w_1, \ldots, w_k, v \) for which \( v \) is a regular node. We have shown above that \( \mu(u) = u \) and \( \mu(v) = v \) must hold in all \( \mu \). The nodes \( w_1, \ldots, w_k \) must be mapped to intermediate nodes on the path from \( u \) to \( v \) in \( D \). There are \( \theta(u, v) \) intermediate nodes on this path.

- If the core path \( u, w_1, \ldots, w_k, v \) contains only child edges, then \( \theta(u, v) = k \), since \( \theta \) is path respecting. Therefore, there is only a single way to map \( w_1, \ldots, w_k \) to database nodes, i.e., the number of choices of mappings equals \( b_{u,v} \).

- Suppose now that the core path has \( d_{u,v} \) descendant edges and \( c_{u,v} \) child edges (where \( d_{u,v} \geq 1 \)). Since \( \theta \) is path respecting, \( \theta(u, v) \geq k \). Since queries that are flip-isomorphic are equivalent, we may assume, without loss of generality that the first \( c_{u,v} \) edges are child edges, and the remaining \( d_{u,v} \) edges are descendant edges. Therefore, every matching \( \mu \) must map \( w_1, \ldots, w_{c_{u,v}} \) to the first \( c_{u,v} \) intermediate nodes on the path from \( u \) to \( v \) in \( D \). There are \( d_{u,v} - 1 \) nodes on the path \( w_{c_{u,v}+1}, \ldots, w_k \) (since the last descendant edge is the incoming edge for node \( v \)). We must choose mappings for these nodes from the remaining \( \theta(u, v) - c_{u,v} \) intermediate nodes on the path from \( u \) to \( v \) in \( D \). There are in total

\[
b_{u,v} = \binom{\theta(u, v) - c_{u,v}}{d_{u,v} - 1}
\]

such choices.
Now, consider a core path \( u, w_1, \ldots, w_k, v \) for which \( v \) is a flexible node. We have shown above that \( \mu(u) = u \). However, \( \mu(v) \) is not necessarily equal to \( v \). Since \( v \) is a leaf node, the nodes on the path \( w_1, \ldots, w_k, v \) must be mapped to nodes on the path from \( u \) to \( v \) in \( D \) (because there are no nodes below \( v \) in \( D \)). We count the number of different ways to map the nodes \( w_1, \ldots, w_k, v \). The reasoning is similar to that given above, and is presented for completeness.

- If the core path \( u, w_1, \ldots, w_k, v \) contains only child edges, then \( \theta(u, v) \geq k \), since \( \theta \) is path respecting. However, there is still only a single way to map \( w_1, \ldots, w_k, v \) to database nodes, i.e., the number of choices of mappings equals \( b_{u,v} \).

- Suppose now that the core path has \( d_{u,v} \) descendant edges and \( c_{u,v} \) child edges (where \( d_{u,v} \geq 1 \)). Since \( \theta \) is path respecting, \( \theta(u, v) \geq k \). Since queries that are flip-isomorphic are equivalent, we may assume, without loss of generality that the first \( c_{u,v} \) edges are child edges, and the remaining \( d_{u,v} \) edges are descendant edges. Therefore, every matching \( \mu \) must map \( w_1, \ldots, w_{c_{u,v}} \) to the first \( c_{u,v} \) intermediate nodes on the path from \( u \) to \( v \) in \( D \). There are \( d_{u,v} \) nodes on the path \( w_{c_{u,v}+1}, \ldots, w_k, v \) (since we are including \( v \) in this path). We must choose mappings for these nodes from the remaining \( \theta(u, v) - c_{u,v} + 1 \) nodes on the path from \( u \) to \( v \) (including \( v \)) in \( D \). There are in total

\[
b_{u,v} = \binom{\theta(u, v) - c_{u,v} + 1}{d_{u,v}}
\]

such choices.

Since the choices for the nodes on the core paths can be made independently, this concludes the proof.

### 4.4 Equivalence Characterization

We use \( \max_{cp}(Q) \) to denote the number of intermediate nodes on the longest core path in \( Q \). Similarly, for a multiset of queries \( \mathcal{Q} \), we define

\[
\max_{cp}(\mathcal{Q}) = \max\{ \max_{cp}(Q) \mid Q \in \mathcal{Q} \}
\]

For example, consider queries \( Q \) and \( Q' \) in Figure 6. Then, \( \max_{cp}(Q) = 1 \) (due to the core path \( u, y, v \)) and \( \max_{cp}(Q') = 2 \) (due to the core path \( u', y'_1, y'_2, x' \)).

We now state and prove the main result of this article. The basic strategy in our proof is to identify a query for which a family of canonical databases may be created. We then show that if \( Q \) and \( Q' \) do not satisfy a certain property, then the number of times that \( Q \) and \( Q' \) return the target series over the canonical databases are different polynomials. This proof is somewhat in the spirit of the proofs used to characterize bag equivalence of conjunctive Datalog queries with or without comparisons [6] but is significantly more intricate due to the presence of recursion, implied by the descendant edges.
Theorem 4.11 Let \( Q \) and \( Q' \) be multisets of completely ordered queries. Let 
\[ k = \max_{cp}(Q \cup Q') + 1. \]
Then, 
\[ Q \equiv Q' \iff Q \sim_k Q'. \]

Proof. It follows from Propositions 4.2 and 4.5 that if \( Q \sim_k Q' \), then also \( Q \equiv Q' \). Thus, it remains to show the other direction. In fact, we show the contrapositive, i.e., that if \( Q \not\sim_k Q' \), then also \( Q \not\equiv Q' \).

We may assume that \( Q \) and \( Q' \) are \( k \)-unrolled (otherwise we compute these unrollings). We may assume that there is no pair of queries \( Q \in Q \) and \( Q' \in Q' \) such that \( Q \sim Q' \). Otherwise, such queries always contribute the same results with the same multiplicities to \( Q \) and \( Q' \), and can be removed. Since \( Q \not\sim_k Q' \), some queries remain.

Among the queries in \( Q \) and in \( Q' \), we find a query \( Q_1 \) that is minimal in its number of core nodes, and among those, minimal in its number of non-wildcard labeled core nodes (i.e., maximal in its number of wildcard nodes). We will be creating a family of canonical databases for this query and will show that over some database in this family \( Q \) and \( Q' \) return the target series a different number of times.

By Proposition 4.9 and our choice of \( Q_1 \), it follows that any query in \( Q \) or \( Q' \) that returns the target series over a canonical database for \( Q_1 \) must be core isomorphic to \( Q_1 \). Thus, we can assume that all queries in \( Q \) and \( Q' \) are core isomorphic to \( Q_1 \), since all other queries will not return the target series over the family of database that we define.

Not only do we wish to create canonical databases, we wish to make these databases strictly path respecting for some query in \( Q \) or \( Q' \). To choose this query, we find among all those (core-isomorphic) queries of \( Q \) and \( Q' \), a query \( Q_2 \) for which the following property holds: There is no \( Q \in Q \cup Q' \), such that for all pairs of core endpoints \( u, v \) in \( Q \) is shorter than or equal to the core path from \( u \) to \( v \) in \( Q_2 \), and there exist core endpoints \( u', v' \) such that the core path from \( u' \) to \( v' \) in \( Q \) is shorter than the core path from \( u' \) to \( v' \) in \( Q_2 \). Clearly, such a query \( Q_2 \) exists.

Now, our family of databases is defined to be canonical databases \( D_{Q_2}^\theta \) such that \( \theta \) is strictly path-respecting for \( Q_2 \). Since all queries in \( Q \) and \( Q' \) have isomorphic cores, only queries for which \( \theta \) is path respecting can return the target series. Suppose that there is a query \( Q \in Q \cup Q' \) for which there is a pair of core endpoints \( u, v \) in which the core path from \( u \) to \( v \) in \( Q \) is longer than that of \( Q_2 \). Since only paths of maximal length \( k \) can have descendant edges, we can conclude that the path from \( u \) to \( v \) in \( Q_2 \) contains only child edges. Therefore, \( \theta \) cannot be path-respecting for \( Q \), as it must contain at most as many edges as there are from \( u \) to \( v \) in \( Q_2 \). Thus, we can assume that such queries \( Q' \) do not exist in \( Q \) or \( Q' \), i.e., all queries in \( Q \) and \( Q' \) have core paths of precisely the same lengths as the corresponding core paths in \( Q_2 \). (Shorter core paths are not possible by our choice of \( Q_2 \).) There may be core paths of length \( k \) that vary in their number of descendant edges, however.

Finally, we now show that given such a canonical database \( D_{Q_2}^\theta \), the number
of times that $Q$ and $Q'$ return the target series are different polynomials in the values assigned by $\theta$. By Proposition 4.10, $Q$ and $Q'$ return the target series
\[ \sum_{Q \in Q} \Phi(Q, \theta) \quad \text{and} \quad \sum_{Q' \in Q'} \Phi(Q', \theta) \] times, respectively.

Clearly, these summations in Equation 4 are polynomials in the values that $\theta$ assigns. It remains to be shown that they are different. Specifically, we will show that there is a monomial that appears in one of these polynomials, but not in the other. To find this monomial, we identify a query $Q_3$ such that for every other (non-isomorphic) query $Q \in Q \cup Q'$, there is at least one pair of core endpoints $u, v$, for which the path from $u$ to $v$ in $Q$ has more descendant edges than the path from $u$ to $v$ in $Q_3$. Such a query $Q_3$ must obviously exist. Assume, without loss of generality, that $Q_3 \in Q$.

Let $(u_1, v_1), \ldots, (u_l, v_l)$ be all pairs of core endpoints in $Q_3$ whose paths contain descendant edges. We assume that there is no $v_i$ that is a leaf node, labeled $\ast$, and not in $\bar{o}$. If such a $v_i$ does exist, a similar argument to the one presented applies.

By Proposition 4.10, the monomial
\[ \prod_{i=1}^l \frac{1}{d_{u_i, v_i}} (\theta(u_i, v_i))^{d_{u_i, v_i}} \]
is in $\Phi(Q, \theta)$. By our assumption that there are no flip-isomorphic queries in $Q'$ (as pairs of flip-isomorphic queries in $Q$ and $Q'$ were removed earlier) this monomial cannot appear in $\Phi(Q', \theta)$ for any $Q' \in Q'$. We may conclude that
\[ \sum_{Q \in Q} \Phi(Q, \theta) \neq \sum_{Q' \in Q'} \Phi(Q', \theta), \]
as required.\(^7\) This concludes the proof.

From the proof of this theorem, we can derive the following corollary, which will be useful in Section 5. Given a multiset of queries $Q$ and a query $Q$, we use $Q|Q$ to denote the subset of queries $Q'$ in $Q$ that are core isomorphic to $Q$.

**Corollary 4.12** Let $Q$ and $Q'$ be multisets of completely ordered queries. Then $Q \not\equiv Q'$ if and only if there exists a query $Q \in Q \cup Q'$ and a path respecting mapping $\theta$ such that
\[ Q|Q(D_Q^\theta) \neq Q'|Q(D_Q^\theta). \]

Taken together, Theorem 4.11 and Corollary 3.6 provide an equivalence characterization for arbitrary multisets of queries (that may not be completely ordered).

\(^7\)Note that although the choice of $\theta$ is limited by requiring $\theta$ to be path respecting, there are still choices of $\theta$ which will yield the inequality.
Corollary 4.13 Let $Q$ and $Q'$ be multisets of queries, which may not be completely ordered. Then,

$$Q \equiv Q' \iff \text{Exp}(Q) \sim^f_k \text{Exp}(Q'),$$

where $k = \max_{cp}(\text{Exp}(Q \cup Q')) + 1$.

Based on Corollary 4.13, we present an upper bound on the complexity of equivalence. The following result relies on the facts that (1) the size of every query in the multiset $\text{Unroll}_k(\text{Exp}(Q))$, $\text{Unroll}_k(\text{Exp}(Q'))$ is bound by a polynomial in the size of the input and (2) using nested loops, we can check for the equivalence characterization, without generating all queries $\text{Unroll}_k(\text{Exp}(Q))$, $\text{Unroll}_k(\text{Exp}(Q'))$ at the same time.

Theorem 4.14 Let $Q$ and $Q'$ be multisets of queries, which may not be completely ordered. It is possible to determine whether $Q \equiv Q'$ in PSPACE, in the size of $Q \cup Q'$.

Proof. Let $Q$ be a query with $n$ nodes. Obviously, the number of queries in $\text{Unroll}_k(\text{Exp}(Q))$ may be exponential in the size of $Q$ and in $k$. However, we start by showing that the number of nodes in a query of $\text{Unroll}_k(\text{Exp}(Q))$ is bounded by a polynomial in $k$ and $n$.

Let $Q'$ be a query in $\text{Exp}(Q)$. While expanding a pair of nodes $v, w$ in $Q$, the number of nodes may decrease (when $v$ and $w$ are merged) or increase (as a new node is created when $v$ and $w$ are both lowered). By Theorem 3.5, $Q'$ will have at most $2n$ nodes.

Let $Q''$ be a query in $\text{Unroll}_k(Q')$. We have established that $Q'$ has at most $2n$ nodes. In the worst case, we lengthen the path between every pair of adjacent nodes in $Q'$ by adding $k$ intermediate nodes. Since $Q'$ is a tree, there are at most $2n - 1$ pairs of adjacent nodes. Thus, $Q''$ will have at most $(2n - 1)(k + 1) + 1$ nodes. Thus, we have shown that every query in $\text{Unroll}_k(\text{Exp}(Q))$ has at most $(2n - 1)(k + 1) + 1$ nodes.

Now, let $Q$ and $Q'$ be multisets of queries. We show how to determine whether $Q \equiv Q'$ in polynomial space. We start by iterating over the queries in $\text{Unroll}_k(\text{Exp}(Q))$. At any moment in the iteration we store only the single query at hand, as well as the choices made to create this query. By the result established above, this requires only polynomial space. For each query $Q \in \text{Unroll}_k(\text{Exp}(Q))$, we perform two inner loops. First, we iterate over all queries in $\text{Unroll}_k(\text{Exp}(Q))$ and count the number of queries that are flip isomorphic to $Q$. (Determining whether two queries are flip isomorphic can be done in linear time by a simple tree traversal. Storing this number will take polynomial space, as there are at most exponentially many flip isomorphic queries.) Second, we iterate over all queries in $\text{Unroll}_k(\text{Exp}(Q'))$ and count the number of queries that are flip isomorphic to $Q$. If these numbers do not match, we have established that $Q \not\equiv Q'$. Otherwise, once we finish iterating over $\text{Unroll}_k(\text{Exp}(Q))$, we repeat the process, but now iterate over $\text{Unroll}_k(\text{Exp}(Q'))$ in the outer loop. If we complete all iterations without establishing nonequivalence, we have determined that $Q \equiv Q'$. 26
Since our iterations are always over objects that require polynomial space, the entire process requires only polynomial space.

We now consider the containment problem. As is the case for Datalog queries, containment is significantly more difficult than equivalence. In fact, even when \( Q \) and \( Q' \) are orderless, determining whether \( Q \subseteq Q' \) is undecidable. The proof of Theorem 4.15 is in the spirit of a similar result for bag containment of unions of Datalog queries in [13].

**Theorem 4.15** The problem of deciding bag containment among orderless multisets of queries is undecidable.

**Proof.** Let \( \Psi(x_1, \ldots, x_k) \) and \( \Psi'(x_1, \ldots, x_k) \) be polynomials in \( k \) variables with positive integer coefficients and no constant terms. The problem of determining whether

\[
\Psi(x_1, \ldots, x_k) \leq \Psi'(x_1, \ldots, x_k)
\]

for all nonnegative integers \( x_1, \ldots, x_k \) is known to be undecidable. (See [13]. This follows from the undecidability of determining existence of a nonnegative integer solution to a Diophantine equation [9].)

Given polynomials \( \Psi \) and \( \Psi' \), we construct multisets of orderless queries \( Q \) and \( Q' \) such that \( Q \subseteq Q' \) if and only if Equation 5 is true, for all nonnegative integers \( x_1, \ldots, x_k \).

Suppose that \( \Psi \) is of the form

\[
\Psi(x_1, \ldots, x_k) = \sum_{i=1}^{n} a_i x_1^{c_{i,1}} \cdots x_k^{c_{i,k}}
\]

For each \( i \leq n \), we create a query \( Q_i \), which has the following structure:

- The root of \( Q_i \) is labeled \( R \), and is the only output node.
- For all \( j \leq k \), the root has \( c_{i,j} \) children labeled \( X_j \).

Now, the multiset \( Q \) has \( a_i \) copies of \( Q_i \), for all \( i \leq n \). The multiset \( Q' \) is defined similarly.

We now show that \( \Psi(x_1, \ldots, x_k) \leq \Psi'(x_1, \ldots, x_k) \) if and only if \( Q \subseteq Q' \). First, suppose that \( Q \subseteq Q' \). Let \( v_1, \ldots, v_k \) be an assignment of nonnegative integers for \( x_1, \ldots, x_k \). Let \( D \) be the database that has a root labeled \( R \), and \( v_j \) children labeled \( X_j \), for all \( j \leq k \). It is easy to see that \( Q \) (resp. \( Q' \)) returns the root of \( D \) exactly \( \Psi(v_1, \ldots, v_k) \) (resp. \( \Psi'(v_1, \ldots, v_k) \)) times. Hence, \( Q \subseteq Q' \) implies that \( \Psi(v_1, \ldots, v_k) \leq \Psi'(v_1, \ldots, v_k) \).

Now, suppose that \( \Psi(x_1, \ldots, x_k) \leq \Psi'(x_1, \ldots, x_k) \). Let \( D \) be an arbitrary database. If \( D \) is not rooted at a node labeled \( R \), then clearly neither \( Q \) nor \( Q' \) will return any results, hence \( Q(D) \) is a subbag of \( Q'(D) \). Similarly, since the coefficients are positive, neither \( Q \) nor \( Q' \) will return any result if the root of \( D \) does not have at least one child \( X_j \) for all \( j \leq k \). Hence, again \( Q(D) \) is
a subbag of $Q'(D)$. Assume that $D$ does have a root labeled $R$ with at least one child labeled $X_j$ for all $j \leq k$. Clearly any nodes other than these children labeled $X_j$ for $j \leq k$ will not effect the result of either query. Let $v_j$ be the number of children of the root labeled $X_j$. It is also easy to see that $Q$ (resp. $Q'$) will return the root of $D$ exactly $\Psi(v_1,\ldots,v_k)$ (resp. $\Psi'(v_1,\ldots,v_k)$) times. Hence, $\Psi(v_1,\ldots,v_k) \leq \Psi'(v_1,\ldots,v_k)$ implies that $Q(D)$ is a subbag of $Q'(D)$.

The problem of determining whether Equation 5 always holds is equivalent to the problem of determining containment among multisets of orderless queries. Hence, the latter problem is undecidable, as required.

5 Equivalence Testing Procedure

In the previous section we presented a complete characterization of bag equivalence of bags of queries. Our equivalence characterization immediately gives rise to a three step procedure for testing equivalence of $Q$ and $Q'$:

- **Step 1:** Compute the complete expansions $P = Exp(Q)$ and $P' = Exp(Q')$.
- **Step 2:** For an appropriate value of $k$, find an equivalent $k$-unrolled bag of queries $R$ for $P$ and $R'$ for $P'$.
- **Step 3:** Test whether $R$ and $R'$ are flip isomorphic.

Step 3 can be performed in polynomial time in the size of $R$ and $R'$. However, unfortunately, Steps 1 and 2 can each cause an exponential blowup in the number of queries.

In this section we present an equivalence testing procedure that avoids the exponential blowup of Step 2. In other words, for completely ordered bags of queries $Q$ and $Q'$, this procedure will run in polynomial time. To achieve this goal, we employ a novel reduction of the bag equivalence problem to that of testing for path equivalence of nondeterministic finite automata (NFA), a problem which is known to be in polynomial time [20].

The approach presented in this section is of interest for two reasons. First, as detailed above, it can avoid one of the sources of exponential blowup in testing for equivalence. Second, this proof technique is quite novel and can potentially prove useful for characterizing equivalence of additional types of queries. We note that this section does not yield the previous section unnecessary. On the contrary—our proofs of correctness in this section are based on results shown in the previous section. In addition, our characterization in the previous section presents a structural condition for equivalence, whereas determining equivalence in this section is presented via an algorithm (and reduction).

To make this article self-contained, we start by defining NFA. We then present a representation of databases $D$ as strings $S(D)$ and a representation of queries $Q$ as NFA $N(Q)$. Next we prove a correspondence between the number of accepting paths of a query NFA $N(Q)$ on a string database $S(D)$, and the
multiplicity of results when applying query $Q$ to database $D$. This correspondence will allow us to reduce bag equivalence of queries to path equivalence of NFA. To simplify the presentation we will assume throughout most of this section that all queries are Boolean, i.e., have an empty output tuple. We relax this assumption at the end of the section and show the small changes needed to ensure that the results hold for non-Boolean queries, too.

5.1 Nondeterministic Finite Automata

A nondeterministic automaton (NFA) is a 5-tuple $N = (P, \Sigma, \delta, p_0, F)$, where:
- $P$ is a finite set of states,
- $\Sigma$ is a finite set of symbols,
- $\delta : P \times \Sigma \rightarrow 2^P$ is a transition function,
- $p_0 \in P$ is the initial state, and
- $F \subseteq P$ is a set of accepting states.

Note that empty transitions (i.e., state transitions without an input letter) are not allowed in our automata.

An automaton reads a word $(A_1 \cdots A_n) = \bar{A} \in \Sigma^*$, where each $A_i$ is in $\Sigma$. A run on word $(A_1 \cdots A_n)$ is a sequence of states $\bar{p} = (p_0, p_1, \ldots, p_n) \in P^{n+1}$, which begins with $p_0$, and $p_{i+1} \in \delta(p_i, A_{i+1})$ for all $0 \leq i < n$. We say that a run $\bar{p}$ accepts word $\bar{A}$ if $p_n \in F$. We say that an NFA $N$ accepts word $\bar{A}$, if there is some run $\bar{r}$ which accepts $\bar{A}$. Note that the number of accepting runs for a given word is always finite, since we do not allow empty transitions. We use $|N(\bar{A})|$ to denote the number of accepting runs for $N$ on $\bar{A}$.

Bag equivalence of queries will be reduced to path equivalence of NFA, defined next.

**Definition 5.1 (Path Equivalence)** Let $N_1$ and $N_2$ be NFA. We say that $N_1$ and $N_2$ are path equivalent if for every word $\bar{A} \in \Sigma^*$, the following holds

$$|N_1(\bar{A})| = |N_2(\bar{A})|.$$

5.2 String Representation of a Database

We now present a string representation for databases. Intuitively, to represent a database $D$, we list the nodes of $D$ by a depth-first left-to-right preorder. In order to retain all information about the structure of the database, we use opening brackets to indicate branching at internal nodes, and closing brackets to indicate leaf nodes. Some technical definitions, and then our precise representation, are presented next.

Let $D = (V, E, r, \prec, \lambda)$ be a database. We define the branching level of internal nodes $v \in V$ as one less than the number of children of $v$, and denote
Algorithm DatabaseToString(D)

1: \( S \leftarrow [\) \\
2: \text{RecursiveStrAppend}(D, \text{root}(D), S) \\
3: \text{return } S

Algorithm RecursiveStrAppend(D, a, S)

1: \( S \leftarrow \text{append}(S, \lambda(a)) \) \\
2: \text{if } a \text{ is a leaf then} \\
3: \quad S \leftarrow \text{append}(S,\text{']'}) \\
4: \text{else} \\
5: \quad \text{for } i \leftarrow 1 \text{ to } \text{branch}(a) \text{ do} \\
6: \quad \quad S \leftarrow \text{append}(S,\text{'['])} \\
7: \quad \text{for each child } b \text{ of } a \text{ in order do} \\
8: \quad \quad \text{RecursiveStrAppend}(D, b, S)

Figure 10: Algorithm for deriving string representation of a database.

this number with \( \text{branch}(a) \). Finally, in the following, we use \( x^y \) to denote the string containing \( y \) repetitions of the symbol ‘\( x \)’.

Our algorithm for translating databases to strings appears in Figure 10. It starts by prepending the string ‘\( [\)’. (This is a technical detail useful to ensure that the brackets are balanced). Then, starting at the root \( r \) of \( D \), we traverse \( D \) using a depth-first left-to-right preorder. When we reach a node \( a \), we first append its label \( \lambda(a) \). Next, if \( a \) is a leaf, we indicate this fact by appending a closing bracket ‘\( ]\)’. Otherwise we append ‘\( [\) \( \text{branch}(a) \) times (i.e., as many times as the branching level of \( a \)), and then recurse over the children of \( a \).

Example 5.2 Consider the database depicted in Figure 11. The ordering of the nodes are defined in the left-to-right order in which nodes appear. This database is represented as the string

\[
\]

as explained next:

- We start by prepending the opening bracket ‘\( [\)’.
- Next we add the label of the root \( A \). Since the branching level of \( A \) is one (as \( A \) has two children), we add one ‘\( [\)’.
- We recurse down to \( B \), add its label, and no opening brackets, as \( B \) has branching level of 0. Next, we reach leaf \( C \), whose label is added, along with a closing bracket ‘\( ]\)’. 

30
Figure 11: A database $D$. The string representation of $D$ is presented in Example 5.2.

- We now recurse down to the second child of $A$, namely $C$, add its label and two opening brackets (as $C$ has branching level 2), and continue thus to the children and descendants of $C$.

We denote the string representation of database $D$ as $S(D)$, and we show the following.

**Proposition 5.3** Let $D$ be a database. Then, $S(D)$ is a string with balanced brackets.

**Proof.** Let $D$ be a database and let $W$ and $U$ be the sets of all internal nodes and leaf nodes, respectively, in $D$. Define

$$\text{branch}(D) := \sum_{w \in W} \text{branch}(w)$$

$$\text{leaf}(D) := |U|.$$

The number of closing brackets in $D$ is $\text{leaf}(D)$, while the number of opening brackets in $D$ is $\text{branch}(D) + 1$ (due to the prepending of '['). Hence it is sufficient to show that $\text{branch}(D) + 1 = \text{leaf}(D)$. As we will show that this holds inductively on every subtree of $D$ we will derive that not only are the number of opening and closing brackets equal, but the brackets are in fact balanced.

We show that $\text{branch}(D) + 1 = \text{leaf}(D)$ by induction on the value of $\text{branch}(D)$. For the base case, $\text{branch}(D) = 0$, i.e., no node in $D$ has more than one child. In this case, $D$ consists of a single path, and must, indeed have one leaf, i.e., $\text{leaf}(D) = 1$, and hence, $\text{branch}(D) + 1 = \text{leaf}(D)$.

We now assume correctness for all $\text{branch}(D) < k$, and show for $\text{branch}(D) = k > 0$. Let $v$ be the first node encountered on a depth-first left-to-right preorder starting from the root of $D$ such that $\text{branch}(v) > 0$. (Note that $v$ may in fact be the root.) Let $D_v$ be the subtree of $D$ rooted at $v$. 

31
Let \( u_1, \ldots, u_n \) be the children of \( v \), and denote the subtree rooted at \( u_i \) as \( D_{u_i} \). Clearly,

\[
\text{leaf}(D) = \sum_{i=1}^{n} \text{leaf}(D_{u_i})
\]

\[
\text{branch}(D) = n - 1 + \sum_{i=1}^{n} \text{branch}(D_{u_i}).
\]

By the induction hypothesis, we have \( \text{branch}(D_{u_i}) + 1 = \text{leaf}(D_{u_i}) \), for all \( i \). Hence, we derive

\[
\text{branch}(D) + 1 = n + \sum_{i=1}^{n} \text{branch}(D_{u_i})
\]

\[
= n + \sum_{i=1}^{n} (\text{leaf}(D_{u_i}) - 1)
\]

\[
= \sum_{i=1}^{n} \text{leaf}(D_{u_i})
\]

\[
= \text{leaf}(D)
\]

as required.

In general, we will be interested in strings that represent canonical databases for a given query \( Q \) and mapping \( \theta \). To make our proofs go smoother, we will slightly amend the definition of a canonical database. In particular, recall that during the wildcard elimination step we replaced all wildcards with a single unused label \( Z \). In this section, we will instead use two different unused labels during wildcard elimination. (Recall that this does not affect correctness of any of our results, as explained in Remark 4.8.) To be precise, in this section, when referring to a canonical database, we assume that wildcards are replaced in the following fashion. Let \( v \) be a node labeled \( \ast \).

- If \( v \) is the root or has more than one child then we replace \( \ast \) with unused label \( Y \).
- Otherwise, we replace \( \ast \) with unused label \( Z \).

A final useful definition is presented.

**Definition 5.4 (Minimal Accepting Database)** Let \( Q = (V, E_f \cup E_g, r, \prec, \lambda, \bar{o}) \) be a completely ordered query. Let \( \theta \) be a the path respecting mapping of core paths to values that maps each pair of core endpoints \( u, v \) in \( Q \) to the minimal path respecting value. We say that the canonical database \( D^Q_\theta \) is the minimal accepting database for \( Q \).

Note that \( \theta(u, v) = c_{u,v} + d_{u,v} - 1 \), in a minimal accepting database. Note also that there is precisely one matching for \( Q \) over its minimal accepting database.
Example 5.5 Consider query $Q$ appearing in Figure 12. Two databases (again with ordering implicit in the figure) appear in this figure. Both $D_1$ and $D_2$ are canonical databases. Note the usage of labels $Y$ and $Z$. Database $D_1$ is the minimal accepting database for $Q$.

5.3 NFA Representation of a Query

In this section, we will show how to convert completely ordered queries into NFA. The NFA are created in a special manner, so that runs over the string representation $S(D)$ of a canonical database $D$ for the query will mimic matchings of the query over $D$. Note that we do not define an NFA that mimics a query for all databases, but rather only for canonical databases. The expressive power of NFA does not allow defining such a general automaton.\footnote{For example, NFA cannot even be defined so as to accept only strings with balanced brackets, as can be shown using the Pumping Lemma \[17\].}

Let $Q = (V, E_j \cup E_g, r, \prec, \lambda, \overline{o})$ be a completely ordered query. We convert $Q$ into an NFA $N = (P, \Sigma, \delta, p_0, F)$ using the algorithm appearing in Figure 13. The algorithm \texttt{QueryToNFA} starts by initializing the NFA $N$ (Line 1), with:

- a starting state $p_0$ (which is the only state);
- the alphabet containing all symbols in $Q$ (denoted $\lambda(V)$), as well as four special symbols `$Z$', `$Y$' (used instead of wildcards) and `['', ']';
- no transitions, and no accepting states.

Next, a call is made to the subprocedure \texttt{Extend}. This subprocedure receives the NFA $N$ (being created), a state $p$ and a label $A$, and simply extends $N$ to have an additional state $p'$, with the transition $p, A, p'$. The new state is returned by the subprocedure. Therefore, the call to \texttt{Extend} in Line 2 of \texttt{QueryToNFA} simply adds a new transition for the opening bracket, which is expected to appear at the beginning of every string representation of a database.
Algorithm QueryToNFA(Q)

1: \( N \leftarrow \{p_0\}, \lambda(V) \cup \{Z', Y', [', ']\}, \emptyset, p_0, \emptyset \)
2: \( p \leftarrow \text{Extend}(N, p_0, [') \)
3: \( p \leftarrow \text{RecursiveExtend}(Q, \text{root}(Q), N, p) \)
4: \( N.\text{setAcceptingStates}(\{p\}) \)
5: \( \text{return } N \)

Algorithm RecursiveExtend(Q, v, N, p)

1: \( p \leftarrow \text{Extend}(N, p, \text{Label}(Q, v)) \)
2: if \( v \) is a leaf then
3: if \( \text{Label}(Q, v) = Z' \) then
4: \( N.\text{addTransition}(p, Z', p) \)
5: \( \text{return } \text{Extend}(N, p, [') \)
6: for \( i \leftarrow 1 \) to \( b_v \) do
7: \( p \leftarrow \text{Extend}(N, p, [') \)
8: for each child \( w \) of \( v \) in order do
9: if \((v, w) \in E_p\) then
10: \( N.\text{addTransition}(p, Z', p) \)
11: \( p \leftarrow \text{RecursiveExtend}(Q, w, N, p) \)
12: \( \text{return } p \)

Algorithm Extend(N, p, A)

1: \( p' \leftarrow \text{newState}() \)
2: \( N.\text{addState}(p') \)
3: \( N.\text{addTransition}(p, A, p') \)
4: \( \text{return } p' \)

Algorithm Label(Q, v)

1: if \( \lambda(v) \neq * \) then
2: \( \text{return } \lambda(v) \)
3: else if \( v = \text{root}(Q) \) or \( b_v \geq 1 \) then
4: \( \text{return } Y' \)
5: else
6: \( \text{return } Z' \)

Figure 13: Algorithm for converting ordered queries to NFA.

Next, in Line 3, \texttt{RecursiveExtend} is called, which recursively extends \( N \), adding additional states and transitions, according to the structure of \( Q \). The last state added is returned by the procedure. \texttt{RecursiveExtend} is the heart of the translation process, and will be discussed in detail later on. Finally the
last state added to \( N \) (and returned by \texttt{RecursiveExtend}) is set to be the only accepting state of \( N \), and \( N \) is returned.

Now, we consider \texttt{RecursiveExtend} in detail. This procedure gets, as input, the query \( Q \), a node \( v \) in \( Q \), the NFA \( N \), and a state \( p \) in the NFA. (Typically, this will be the last state added to \( N \).) By closely following the process in which a string representation of a database is created, the goal of \texttt{RecursiveExtend} is to create an NFA that accepts exactly canonical databases. This is achieved in the following fashion:

- We start (Line 1) by extending \( N \) with a transition for the label represented by \( v \). Technically, the procedure \texttt{Label} is used for this purpose—returning \( \lambda(v) \) if that is not a wildcard, ‘Y’ if \( v \) is a root or has more than one child, and ‘Z’ otherwise.

- Next, if \( v \) is a leaf (Lines 2–5), we add a transition for a closing bracket. A string representation of a corresponding canonical database will have a ‘]’ at this point. Note also that if \( v \) is labeled \( \ast \), the algorithm will also add a self-loop (before the ‘]’ transition). This is important in order to ensure that the NFA created accepts all path respecting canonical databases (which may have additional nodes “below” the node corresponding to \( v \)).

- Otherwise, if \( v \) is an internal node (Lines 6–12), we add ‘[’ as many times as \( b_v \) (the branching level of \( v \)). Then, we recursively consider each child \( w \) of \( v \). If \((v, w)\) is a descendant-edge, we add a self loop with the symbol ‘Z’, as descendant edges can be replaced with any length of \( Z \)-labeled paths. Finally, we recursively call \texttt{RecursiveExtend} for \( w \).

We will use \( N(Q) \) to denote the NFA used to represent \( Q \), i.e., the output of \texttt{QueryToNFA}(\( Q \)).

**Example 5.6** Consider query \( Q \) from Figure 12. Figure 14 depicts the corresponding NFA \( N(Q) \) for \( Q \). (Ignore, for now, the variable names appearing below some of the edges.) Note that, when ignoring the self-loops, \( N(Q) \) is simply a single path that accepts the minimal canonical database for \( Q \). The self loops allow \( N(Q) \) to accept all canonical databases for \( Q \). This is, in fact, true for all queries \( Q \) and corresponding NFA \( N(Q) \), as shown next.

**Proposition 5.7** Let \( Q \) be a query and \( N(Q) \) be its query NFA. Then, string \( S \) is accepted by \( N(Q) \) if and only if there is a path-respecting \( \theta \), such that \( S \) is precisely the string representation of \( D^\theta_Q \), i.e., \( S = S(D^\theta_Q) \).

**Proof.** The required follows immediately from the construction of \( N = NFA(Q) \). Observe that:

- Property 1: All cycles are self-loops labeled ‘Z’.

- Property 2: Excluding self-loop edges, \( N \) forms a path from the initial state to the single accepting state.
From Property 2 we derive that there is a single shortest accepted word $S$. By comparing \texttt{QueryToNFA} and \texttt{DatabaseToString} we immediately derive that $S$ is the string representation of the minimal accepting database for $Q$.

From Property 1 we derive that all accepted words can be formed by adding ‘$Z$’ symbols to $S$ at specific places. Note that the ‘$Z$’ symbols can be added precisely at the points in $S$ corresponding to descendant edges in $Q$, as well as to wildcard-labeled leaf nodes. These added ‘$Z$’ symbols correspond to a path of ‘$Z$’ nodes in a canonical database. The fact that ‘$Z$’ are added only at points corresponding to descendant edges ensures that only string representations of \textit{path respecting} canonical databases of $Q$ will be accepted.

Observe that each transition in an NFA belongs to one of the following classes:

- \textit{bracket transitions} having the form $(q_i, A, q_{i+1})$ where $A \in \{[',']\}$;
- \textit{self-loops} having the form $(q_i, Z, q_i)$;
- \textit{label transitions} having the form $(q_i, A, q_{i+1})$ where $A \notin \{[',']\}$.

Each label transition is added at Line 1 of \texttt{RecursiveExtend}. Let $v$ be the node input of \texttt{RecursiveExtend} when the label transition $(q_i, A, q_{i+1})$ is added. Then, we will say that this is the label transition of $v$ in the NFA. Note that there is a bijection between nodes $v$ in $Q$ and label transitions in the NFA, using the correspondence above. We will utilize this bijection in our next result.

\textbf{Example 5.8} Consider again the NFA in Figure 14. Observe that all label transitions are labeled (below the edge) with their corresponding query node. \hfill \Box

We now state and prove the precise correspondence between accepting runs and query matchings.

\textbf{Proposition 5.9} Let $Q$ be a query and let $D$ be a path respecting canonical database for $Q$. Then,

$$|Q(D)| = |N(Q)(S(D))|.$$ 

\textit{Proof.} We show that there is a one to one correspondence between matchings of $Q$ to $D$ and runs of $N(Q)$ on $S(D)$. To this end, we define a function $\pi$...
mapping accepting runs of \(N(Q)\) on \(S(D)\) to matchings of \(Q\) on \(D\). We will show that \(\pi\) is a bijection.

Let \(\vec{r} = r_0, \ldots, r_n\) be an accepting run of \(N(Q)\) on \(S(D) = A_1 \cdots A_n\).\(^9\) Obviously, there is a one to one correspondence between symbols in \(S(D)\) that are not brackets, and nodes in \(D\). Given a non-bracket symbol \(A_i\), we will use \(a_i\) to denote the corresponding database node.

We now define the mapping \(\mu\) from the nodes of \(Q\) to those of \(D\) as follows. Let \(v\) be a node in \(Q\). Let \((p_i, A_i, p_i+1)\) be the label transition corresponding to \(v\). Since \(\vec{r}\) is an accepting run, there must be an index \(j\) such that \(r_j = p_i\) and \(r_{j+1} = p_{i+1}\). This transition occurs when the symbol \(A_{j+1}\) is read. We define \(\mu\) as mapping \(v\) to \(a_{i+1}\) (i.e., the database node corresponding to the symbol which caused the label transition of \(v\) to take place). It is easy to see that \(\mu\) is a matching as the structure of \(S(D)\) mimics that of \(D\) and the run of \(N(Q)\) on \(S(D)\) ensures that all structural requirements of \(Q\) will be satisfied.

We define \(\pi\) as mapping each run \(\vec{r}\) to the corresponding matching \(\mu\), as described above. We must show that \(\pi\) is both injective and surjective. To show that \(\pi\) is injective, observe that every two different runs must differ on at least one symbol in which they perform a label transition. This holds since the symbols of label transitions completely determine all other transitions of the NFA (as the remainder of the NFA is deterministic). To show that \(\pi\) is surjective, observe that every matching can be simulated precisely by an accepting run. \(\Box\)

Now, given a multiset of queries \(Q\), we use \(N(Q)\) to denote the NFA derived by merging the starting states of all NFA \(\{N(Q) \mid Q \in N(Q)\}\). The following corollary follows from Propositions 5.7 and 5.9. Recall that \(Q|_Q\) denotes the subset of queries \(Q'\) in \(Q\) that are core isomorphic to \(Q\).

**Corollary 5.10** Let \(Q\) be a multiset of completely ordered queries. Let \(Q\) be a query and \(\theta\) be a path respecting mapping for \(Q\). Then,

\[
|Q|_Q(D^Q_Q)| = |N(Q)(S(D^Q_Q))|.
\]

Our main result of this section is now easily shown.

**Theorem 5.11** Let \(Q\) and \(Q'\) be multisets of completely ordered queries. Then \(Q \equiv Q'\) if and only if \(N(Q)\) and \(N(Q')\) are path equivalent.

**Proof.** If \(Q \neq Q'\), then by Corollary 4.12, there is a query \(Q \in Q \cup Q'\) and a path respecting \(\theta\) such that \(Q|_Q(D^Q_Q) \neq Q'|_Q(D^Q_Q)\). (Note that \(Q|_Q(D^Q_Q) \neq Q'|_Q(D^Q_Q)\) implies that \(|Q|_Q(D^Q_Q) \neq |Q'|_Q(D^Q_Q)|\), since the queries are Boolean.) By Corollary 5.10, \(S(D^Q_Q)\) is a counter example for path equivalence of \(N(Q)\) and \(N(Q')\).

If \(N(Q)\) and \(N(Q')\) are not path equivalent, then there is some string \(S\) for which \(|N(Q)(S)| \neq |N(Q')(S)|\). Since, \(N(Q)\) (resp. \(N(Q')\)) only accepts strings

\(^9\)We use \(r_i\) to denote the states in the run, to differentiate these from the names of the states in the NFA, as the same state can appear several times in a run.
representing canonical databases for queries in \( Q \) (resp. \( Q' \)), we can again apply Corollaries 5.10 and 4.12 to derive that \( Q \not\equiv Q' \).

We derive the following corollary, which follows from Theorem 5.11 and from the fact that testing path equivalence of NFA (not containing cycles of empty transitions) is in polynomial time [20].

**Corollary 5.12** Let \( Q_1 \) and \( Q_2 \) be multisets of completely ordered queries. Then \( Q_1 \equiv Q_2 \) can be tested in polynomial time.

### 5.4 Queries with Output Nodes

Throughout this section, we assumed that there are no output nodes, in order to simplify the discussion. It is quite easy to extend our results to the non-Boolean case. Corollary 4.12 is easily strengthened to state the following. (Again, this follows directly from the proof of Theorem 4.11.)

**Corollary 5.13** Let \( Q \) and \( Q' \) be multisets of completely ordered queries. Then \( Q \not\equiv Q' \) if and only if there exists a query \( Q \in Q \cup Q' \) and a path respecting mapping \( \theta \) such that the number of times that \( Q|_Q \) and \( Q'|_Q \) return the target series over \( D_Q^\theta \) differs.

We extend the notion of a string representation of a database \( D \) to that of a string representation of a pair \( S(D, \bar{a}) \), where \( a \) is a tuple of nodes from \( D \). This new type of string representation simply appends a special unused symbol \( O_i \) next to the label of the \( i \)-th node in \( \bar{a} \) (for all \( i \)). We will represent canonical databases \( D_Q^\theta \) as the string for the pair \( D_Q^\theta, \bar{o} \) where \( \bar{o} \) is the target series for \( Q \). (An additional needed detail is that output nodes labeled \( * \) are replaced with \( Y \) in a canonical database.)

We also adapt our NFA \( N(Q) \), by adding a transition for the symbol \( O_i \) immediately after the label transition of the \( i \)-th node in the output tuple of \( Q \). These changes ensure that the number of accepting runs of \( N(Q) \) on \( S(D_Q^\theta, \bar{o}) \) will be precisely the number of matchings of \( Q \) to \( D_Q^\theta \) in which the output nodes are mapped to the target series. Using Corollary 5.13, all other proofs can proceed as is.

### 6 Horizontal Axes

Until now, we considered queries which use the \( < \) relationship among sibling nodes as a horizontal axis. As noted earlier, \( < \) is similar to the XPath following axis, but does not coincide precisely with this axis. In this section we consider several horizontal axes available in XPath, and show how our results can be generalized to allow for these additional axes. To simplify the presentation, we show how to add each of these axes separately to individual queries. The extension to multisets of queries, with several of the different axis types considered, is straightforward.
Note that in each subsection a new notion of a query is defined, and is considered throughout that subsection. To differentiate the queries considered within these sections with the original notion of Definition 2.2, we will call the latter standard queries.

6.1 Relationships among Database Nodes
We start by defining several relationships among pairs of nodes. Let $D$ be a database and $a, b$ be nodes in $D$. We say that $b$ follows $a$ if a depth-first traversal of $D$ reaches $a$ before reaching $b$, and moreover, $b$ is not in the subtree rooted at $a$. Equivalently, $b$ follows $a$ if there exists a node $c$ such that $b$ $c$-follows $a$. We say that $b$ is a following sibling of $a$ if $a$ and $b$ are sibling nodes and $a \prec_D b$ (where $\prec_D$ is the ordering over sibling nodes in $D$). Finally, we say that $b$ is the first-child of $a$ if $b$ is a child of $a$, and there is no child $c$ of $a$ for which $c \prec_D b$. The relationship last-child is defined analogously.

6.2 Following and Following-Sibling Axes
We start by considering queries which may use the following and following sibling axes. Thus, our queries are of the form $Q = (V, E, r, \prec, \prec_f, \prec_{fs}, \lambda, \bar{o})$, where $\prec_f$ and $\prec_{fs}$ are partial orderings representing the following and following-sibling relationships, respectively.

To define query semantics, we extend the notion of a matching in the natural fashion, i.e., a mapping $\mu$ of nodes in $Q$ to those in a database $D$ is a matching of $Q$ to $D$ if it satisfies all conditions in Definition 2.3, and moreover,

- For any pair of nodes $v, w$ such that $v \prec_f w$ it holds that $\mu w$ follows $\mu v$;
- For any pair of nodes $v, w$ such that $v \prec_{fs} w$ it holds that $\mu w$ is a following sibling of $\mu v$.

Remark 6.1 We note the translation of XPath queries with horizontal axes into tree-like patterns is not immediate, e.g., in /a/child::b/following::c there is no indication as to which node is the parent of $c$. However, this is easily dealt with by adding descendant edges from the root of the path to nodes with no apparent parent, such as $c$. The following relationship is then expressed using $\prec_f$, and not with the tree structure.

Equivalence among queries (or multisets of queries) with the following and following-sibling axes is defined in the natural way. The following result shows that adding $\prec_f$ and $\prec_{fs}$ does not increase the expressive power of the query language.

Proposition 6.2 Let $Q = (V, E, r, \prec, \prec_f, \prec_{fs}, \lambda, \bar{o})$ be a query. Then, there exists a multiset of standard queries $Q'$ (i.e., such that each $Q' \in Q$ has empty $\prec_f$ and $\prec_{fs}$ relationships), for which $Q \equiv Q'$. 

39
Proof. Let $Q_1$ be the multiset $Exp(Q)$, where $Exp(Q)$ is computed in the manner described in this article, i.e., ignoring $\prec_f$ and $\prec_{fs}$. For a query $Q' \in Q_1$, and a node $u$ in $Q$, we use $u_{Q'}$ to denote the node in $Q'$ corresponding to $u$ in $Q$. Specifically, if $u$ is a node in $Q'$, then $u_{Q'} = u$. Otherwise, $u$ was merged with additional nodes in the expansion process, while generating $Q'$. Recall that such merges create nodes defined as sets of the nodes merged (see step merging nodes in the expansion process). Thus, $u_{Q'}$ is the node which contains $u$.

Now, for all $v, w$ such that $v \prec_f w$, we eliminate from $Q_1$ all queries $Q'$ in which $w_{Q'}$ does not follow $v_{Q'}$. Similarly, for all $v, w$ such that $v \prec_{fs} w$, we eliminate from $Q_1$ all queries $Q'$ in which $w_{Q'}$ is not a following sibling of $v_{Q'}$. Let $Q_2$ be the result of these eliminations.

Next, for all $Q' \in Q_2$, and for all $v \prec_{fs} w$, we replace incoming edges to $v$ and $w$ with child edges. To see that this is a correct replacement, recall that if $u$ is the parent of $v$ and $w$ in $Q'$, and $v \prec w$, then for all matchings $\mu$, the node $\mu u$ must be the lowest common ancestor of $\mu v$ and $\mu w$. Since we require $v$ and $w$ to be siblings, the node $\mu u$ should be the parent of $\mu v$ and $\mu w$, and hence we ensure this by our edge replacement. Let $Q_3$ be the result of this process.

Finally, let $Q$ be the result of taking all queries from $Q_3$, but dropping their relationships $\prec_f$ and $\prec_{fs}$. Now, it is easy to show that for any database $D$, there is a one to one correspondence between matchings of $Q$ to $D$ and those of $Q$ to $D$. \hfill $\Box$

While the number of queries in $Q$ may be exponential in the size of $Q$ (since the number of queries in $Exp(Q)$ may be exponential in $Q$), as before, these queries can be enumerated in polynomial space. We immediately derive the following corollary.

**Corollary 6.3** Let $Q$ and $Q'$ be multisets of queries, which may not be completely ordered, and may use the following and following sibling axes. It is possible to determine whether $Q \equiv Q'$ in PSPACE.

### 6.3 First and Last Axes

We now consider queries which may use the first axes. All our results are immediately extendable to the last axes, in a completely analogous manner. Our queries are of the form $Q = (V, E, r, \prec, F, \lambda, \bar{o})$, where $F$ is a subset of nodes of $V$, corresponding to the first-child relationship.

Once again, to define query semantics, we extend the notion of a matching in the natural fashion, i.e., a mapping $\mu$ of nodes in $Q$ to those in a database $D$ is a matching of $Q$ to $D$ if it satisfies all conditions in Definition 2.3, and moreover,

- For any node $v \in F$, if $u$ is the parent of $v$ in $Q$, then $\mu v$ is the first child of $\mu u$ in $D$.

In a query, we disregard the incoming edge types when determining whether a node is a following sibling of another node.
Unlike following and following sibling, the first axis does increase the expressive power of our query language, as we show next.

**Proposition 6.4** There exists a query $Q$ using the first-child relationship, for which there is no multiset of standard queries $Q$ such that $Q \equiv Q$.

**Proof.** Let $Q = (V, E, r, \prec, F, \lambda, \bar{\circ})$ where $V = \{r, v\}$, $E$ contains the single child edge $(r, v)$, $\prec = \emptyset$, $F = \{v\}$, $\lambda(r) = \lambda(v) = A$ and $\bar{\circ} = v$. Let $Q$ be a multiset of standard queries. We will show that $Q \not\equiv Q$.

Let $k$ be the maximum number of children of the root, among all queries in $Q$. Let $D$ be the database that has a root labeled $A$, with $k + 1$ children, all labeled $A$. Clearly, $Q$ will return only the first child among all the children of the root. On the other hand, it is easy to see that for all $Q' \in Q$, since the root in $Q'$ has at most $k$ children, $Q'$ cannot differentiate between the first child of the root in $D$, and the second child. Hence, $Q$ cannot return the same result as $Q$ over $D$.

Due to the above proposition, we cannot show a result similar to Proposition 6.2. Instead, we show the following result.

**Proposition 6.5** Let $Q_1$ and $Q_2$ be queries that may use the first axis. Then, there are multisets of standard queries $Q_1$ and $Q_2$ (i.e., that do not use the first axis) such that $Q_1 \equiv Q_2$ if and only if $Q_1 \equiv Q_2$.

**Proof.** We start by setting $Q_1 = \{Q_1\}$. First, whenever, there is a node $v$ in $F$, we add $v \prec w$ for all other children $w$ of the parent of $v$. Then, while there is a query $Q \in Q_1$ with a non-leaf node $u$ such that the set $F$ of $Q$ does not contain any of the children of $u$, we remove $Q$ from $Q_1$ and add the following queries to $Q_1$:

- We create a new node $z$, (1) add a child edge $(u, z)$, (2) add $z$ to $F$, and (3) add $z \prec v$, for all other children $v$ of $u$.
- For each child $v$ of $u$ such that there is no $w$ with $w \prec v$, we create a query $Q_v$ in which (1) we add $v$ to $F$ and (2) we add $v \prec w$ for all other children $w$ of $u$.

Once this process is complete, we drop the set $F$ from all queries in $Q_1$. Intuitively, in this manner we have made all possible choices of “first children” for each internal node of $Q_1$. The set $Q_2$ is defined similarly.

We now show that $Q_1 \equiv Q_2$ if and only if $Q_1 \equiv Q_2$. Since $Q_1 \equiv Exp(Q_1)$, it is sufficient to show that

$$Q_1 \equiv Q_2 \Leftrightarrow Exp(Q_1) \equiv Exp(Q_2).$$

Suppose first that $Exp(Q_1) \neq Exp(Q_2)$. Then, by the proof of Theorem 4.11, there is a canonical database for some query in $Exp(Q_1) \cup Exp(Q_2)$ that shows this non-equivalence. Our construction of canonical databases is
such that for any query $Q \in Exp(Q_1) \cup Exp(Q_2)$, all matchings $\mu$ of $Q$ to the database map the first child $v$ of any query node $u$, to the first child of $\mu u$. (This is because the branching factor of the canonical databases is identical to the branching factor of the original queries.) It then follows that the same canonical database shows non-equivalence of $Q_1$ and $Q_2$. This can be shown since $Q_1$ and $Q_2$ will have precisely the same matchings as $Q_1$ and $Q_2$, respectively.

Now, suppose that $Exp(Q_1) \equiv Exp(Q_2)$. Then, by Theorem 4.11, there is a flip isomorphism between the unrolled versions of $Exp(Q_1)$ and $Exp(Q_2)$. It is easy to see that this flip isomorphism always maps first children of queries in $Exp(Q_1)$ to first children of queries in $Exp(Q_2)$ (since the queries are completely ordered), and it is not difficult to see that such a flip isomorphism is a sufficient condition for equivalence of $Q_1$ and $Q_2$.

Corollary 6.6 follows.

**Corollary 6.6** Let $Q$ and $Q'$ be multisets of queries, which may not be completely ordered, and may use the first and last child axes. It is possible to determine whether $Q \equiv Q'$ in PSPACE.

7 Conclusion

This article focused on bag equivalence of tree patterns. Previously, containment of tree patterns under set semantics has been studied extensively, e.g., [18, 16, 15, 21, 12]. Some of the methods used in this article are in the spirit of techniques employed in work on set semantics. We briefly review the relationship between our work and that on set semantics.

The notion of adornments for tree patterns was introduced in [15]. Adornments combine the intuition behind flip-isomorphisms and edge unrollings. However, under set semantics, the reasoning is simplified, e.g., the number of descendant edges on a path is not of importance—rather only the (non-)existence of such an edge is important. Our canonical databases are similar to those of [15], but a more complex analysis is needed to show that when our characterization does not hold, the results of two queries have different multiplicities (even though they may return the same results under set semantics). We note that in general, bag and set semantics give rise to quite different issues, e.g., for example bag containment for the rather simple queries considered in this paper is already undecidable. Note also that while non-containment under set semantics implies non-containment (and non-equivalence) under bag semantics, the opposite is not correct. Thus, it is difficult to transfer techniques from one setting to the other. We do believe that our technique of expanding queries to complete order may be useful for studying containment of queries with ordering under set semantics, as a similar notion of an expansion has been proven useful for determining containment (or equivalence) of Datalog queries under both set [14] and bag [6] semantics.
To summarize our main results, in this article we characterized bag equivalence of tree patterns, which are an abstract representation of XPath queries. Our results are general, and allow descendants, wild-cards, branching, unions, multiple output nodes and horizontal axes. We have presented a complete structural characterization of equivalence, as well as provided a more efficient method of testing equivalence, based on the notion of path equivalence of NFA. This article is also the first to consider bag semantics for queries that are recursive (i.e., due to the descendant axis). Preliminary results on bag containment were presented.

For future work, we plan on studying bag-equivalence in the presence of a schema. We also intend to study the combination of set and bag semantics for tree patterns, in the spirit of [5], e.g., to model XPath queries with the count operator over a sub-portion of the query. Finally, bag-containment for XPath queries without union is an interesting open problem. We note that characterizing bag-containment (or even determining decidability) is a long open problem for conjunctive Datalog queries, and thus, bag-containment for XPath may also prove illusive.

References


