Bag Equivalence of XPath Queries

Sara Cohen
The Selim and Rachel Benin School of Engineering and Computer Science
The Hebrew University
Jerusalem 91094, Israel
sara@cs.huji.ac.il

Yaacov Y. Weiss
The Selim and Rachel Benin School of Engineering and Computer Science
The Hebrew University
Jerusalem 91094, Israel
yyweiss@cs.huji.ac.il

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 paper is the first to focus on bag semantics for XPath queries. In particular, the problem of bag-equivalence of a large class of XPath queries (modeled as tree patterns) is explored. The queries can contain unions, branching, label wildcards, the vertical child and descendant axes, the horizontal following, following-sibling and immediately-following sibling axes, as well as positional (i.e., first and last) axes. Equivalence characterizations are provided, and their complexity is analyzed. As the descendent axis involves a recursive relationship, this paper is also the first to address bag equivalence over recursive queries, in any setting.

Categories and Subject Descriptors
H.2.3 [**Database Management**]: Languages—*Query languages*; H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms
Theory

Keywords
XPath, query equivalence, bag semantics, multiset semantics

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 [16], XLink [10] and XPointer [9]. 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 paper we focus on the problem of determining equivalence of XPath queries, under *bag semantics*. Formally, given two XPath queries, the equivalence problem is to determine whether the queries will yield the same results, over any database. Containment and equivalence, under of various fragments of XPath have been studied extensively, e.g., [11, 13–15, 17], 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 [11] and containment in the presence of DTDs was studied in [17]. Containment of XPath queries including branching, wildcard labels and the descendant axis was shown to be Co-NP complete in [13]. Containment in the presence of DTDs, disjunctions and variables (comparisons) was studied in [14]. Finally, containment of XPath 2.0 queries (which can include path intersection, path equality, path complementation, for-loops and transitive closure) was studied in [15]. All the above-mentioned work considers only XPath 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 on the left side of 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.

Each of the three boxes on the right side of the same figure depicts a query (or union of queries). We follow the standard

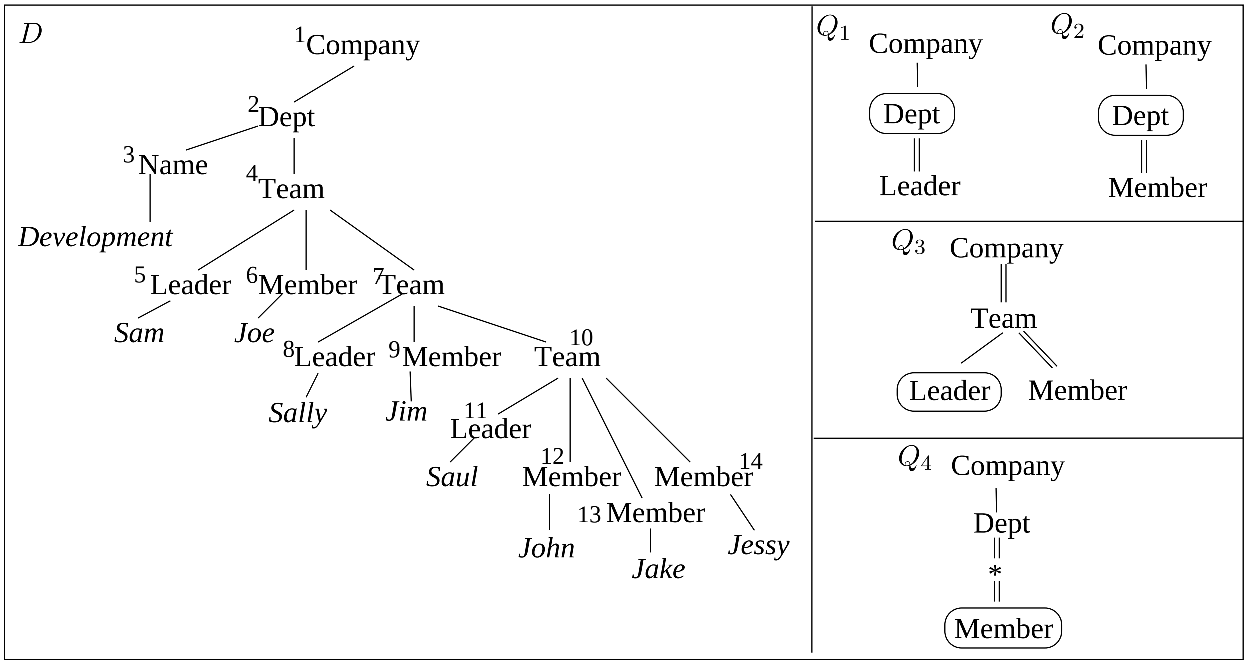

Figure 1: Database and queries. Ovals indicate output nodes.

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 the queries in the top box, i.e., Q_1 and Q_2, will return each department, as many times as it has leaders and members, i.e., the result can be used to count the number of employees per department. The query Q_3 in the middle box returns each leader as many times as the number of (direct and indirect) members within his team. The bottom query Q_4 returns each member, as many times as the number of teams in which the member (directly or indirectly) belongs. $\square$

Equivalence of non-recursive Datalog queries under bag semantics has been studied rather extensively [3,5–7,12] (sometimes in the context of count-queries). However, these results do not carry over to XPath queries, for several reasons. First, XPath queries are evaluated over databases that must have tree form, whereas Datalog queries are evaluated over arbitrary databases. Second, XPath uses 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 XPath queries (as each node has a unique identity in the XML data model), but not for Datalog queries—making the setting quite different. This paper is the first to study equivalence of XPath queries 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 paper imply equivalence characterizations for XPath counting queries.

The main contribution of this paper 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, following-sibling and immediately-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 paper is the first to address bag equivalence over recursive queries, in any setting.

This paper is structured as follows. Section 2 defines the notion of a database, a query and bag semantics for evaluation. To make the presentation clearer, we start by considering only one horizontal axis. Section 3 reduces the equivalence problem to that of equivalence of *completely ordered* sets of queries, and Section 4 provides our equivalence characterization. In Section 5 we show how to extend our results to a richer setting, including different types of horizontal axes. Section 6 concludes.

2. DEFINITIONS

2.1 Databases

Let Σ 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) $\lambda : V \to \Sigma$ 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 is a *non-strict descendant* of a. We say that b *c-follows* a

if a depth-first traversal of D starting at c reaches a, then 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) $a' \prec b'$.

For example, consider the database in Figure 1. Assume that the ordering $\prec$ is defined in the left-to-right order in which nodes appear. Then, Node 7 is an ancestor of Node 9 and a non-strict ancestor of Nodes 7 and 9. Node 12 7-follows Node 8, as does Node 10. However, Node 12 does not 7-follow Node 11.

2.2 Query Syntax

We formally define a query as follows.[1]

Definition 2.1 (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 $E_{/}$ and $E_{//}$, called* child *and* descendant edges*, respectively;*
- *λ is a function $V \rightarrow \Sigma \cup \{*\}$, where $*$ is a special symbol not appearing in Σ, 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 Σ.

If $\prec$ implies a contradiction, then Q is *inconsistent*. Unless otherwise stated, we assume that all queries are consistent in this paper. If $\prec$ is a complete ordering of sibling nodes, Q is a *completely ordered query*. If $\prec$ is empty, then Q is a *orderless query*. We use $\mathcal{Q}$ to denote a *multiset* of queries. If all the queries in $\mathcal{Q}$ are completely ordered (resp. orderless), then we say that $\mathcal{Q}$ is completely ordered (resp. orderless).

2.3 Query Semantics

In this paper, 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 2.2 (Matching). *Let $D = (V_D, E_D, r_D, \prec_D, \lambda_D)$ be a database and $Q = (V_Q, E_{/} \cup E_{//}, r_Q, \prec_Q, \lambda_Q, \bar{o})$ 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: *$\forall (v, w) \in E_{/}$, $(\mu v, \mu w) \in E_D$ and $\forall (v, w) \in E_{//}$, the node μv is an ancestor of μw in D;*
- root consistency: *$\mu r_Q = r_D$;*
- label consistency: *$\forall v \in V_Q$, either $\lambda_Q(v) = *$ 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 μw μu-follows μv, i.e., there exist children a and b of μu in V_D such that*
  1. *a is a non-strict ancestor of μv,*
  2. *b is a non-strict ancestor of μw* and
  3. *$a \prec_D b$.*

[1]The query language is enriched in Section 5.

Note that if Q is completely ordered, then μ must be injective. We use $\mathcal{M}(Q, D)$ to denote the set of all matchings from Q to D.

Remark 2.3. 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 μv and μw be non-strict descendants of *distinct children* a and b of μu. The standard meaning of the following axis would only require μv and μw to have a common ancestor that is a *descendant* of μu, and differs from μv.

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 5. □

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

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

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

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

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 Node 1, and Dept to Node 2. These matchings map Leader to Nodes 5, 8 and 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 Node 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 Nodes 5, 8 and 11 a total of 5, 4 and 3 times, respectively. It is then easy to see that Q_3 returns the each leader exactly as many times as the number of (direct or indirect) 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

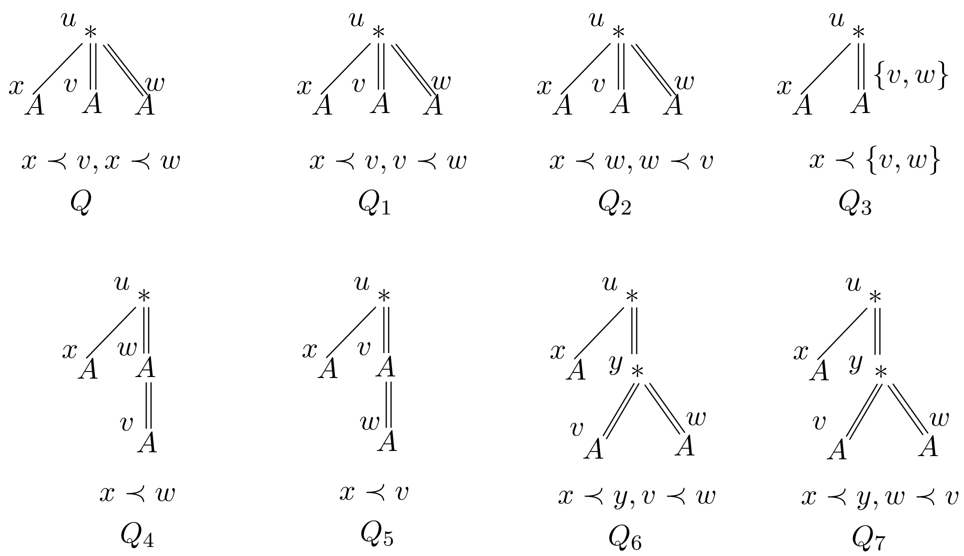

Figure 2: Query Q and $\{v, w\}$-expansion $Exp_{\{v,w\}}(Q) = \{Q_1, \ldots, Q_7\}$.

$*$ 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. □

2.4 Problems of Interest

We say that the multisets of queries $\mathcal{Q}$ and $\mathcal{Q}'$ are *equivalent*, written $\mathcal{Q} \equiv \mathcal{Q}'$, if, for all databases D, it holds that $\mathcal{Q}(D) = \mathcal{Q}'(D)$. Similarly, $\mathcal{Q}$ is contained in $\mathcal{Q}'$, written $\mathcal{Q} \subseteq \mathcal{Q}'$, if, for all databases D, it holds that $\mathcal{Q}(D)$ is a subbag of $\mathcal{Q}'(D)$. 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 paper we focus on the equivalence problem, and provide a complete characterization of equivalence. Limited results are presented on the containment problem (see Section 6 for discussion of the elusiveness of this 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 $\mathcal{Q}$ that is equivalent to Q. Note that care must be taken to ensure that $\mathcal{Q}$ not only returns the same results as Q, but also with the same multiplicities. The key concept used in finding $\mathcal{Q}$ is that of an *expansion*, defined next.

Let $Q = (V, E, r, \prec, \lambda, \bar{o})$ be a query. We say that a pair of nodes v, w are *order-unknown 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 order-unknown sibling nodes, then the $\{v, w\}$-expansion of Q is the multiset of queries $Exp_{\{v,w\}}(Q)$ containing the following queries:

- **Directly Adding Order:** $Exp_{\{v,w\}}(Q)$ contains the queries Q_1 and Q_2, which are derived by adding $v \prec w$ and $w \prec v$, respectively, to Q;
- **Uniting Nodes:** If $\lambda(v) = \lambda(w)$, $\lambda(v) = *$ or $\lambda(w) = *$, then $Exp_{\{v,w\}}(Q)$ 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 $E_{/\!/}$, $E_{/}$, $\prec$, $\bar{o}$ with $\{v, w\}$;
  4. if $(u, \{v, w\})$ is now in both $E_{/\!/}$ and $E_{/}$, then removing $(u, \{v, w\})$ from $E_{/\!/}$.

  In other words, nodes v, w are *united* in Q_3.[2]
- **Demoting a Node:** If $(u, v) \in E_{/\!/}$, then $Exp_{\{v,w\}}(Q)$ 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 *demoted* to become a child of w.

  Similarly, if $(u, w) \in E_{/\!/}$, then $Exp_{\{v,w\}}(Q)$ contains the query Q_5, which is created symmetrically, by demoting w to become a child of v.
- **Lowering Both Nodes:** If $(u, v) \in E_{/\!/}$ and $(u, w) \in E_{/\!/}$, then $Exp_{\{v,w\}}(Q)$ contains the query Q_6, which is derived by
  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 $E_{/\!/}$;

[2]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.

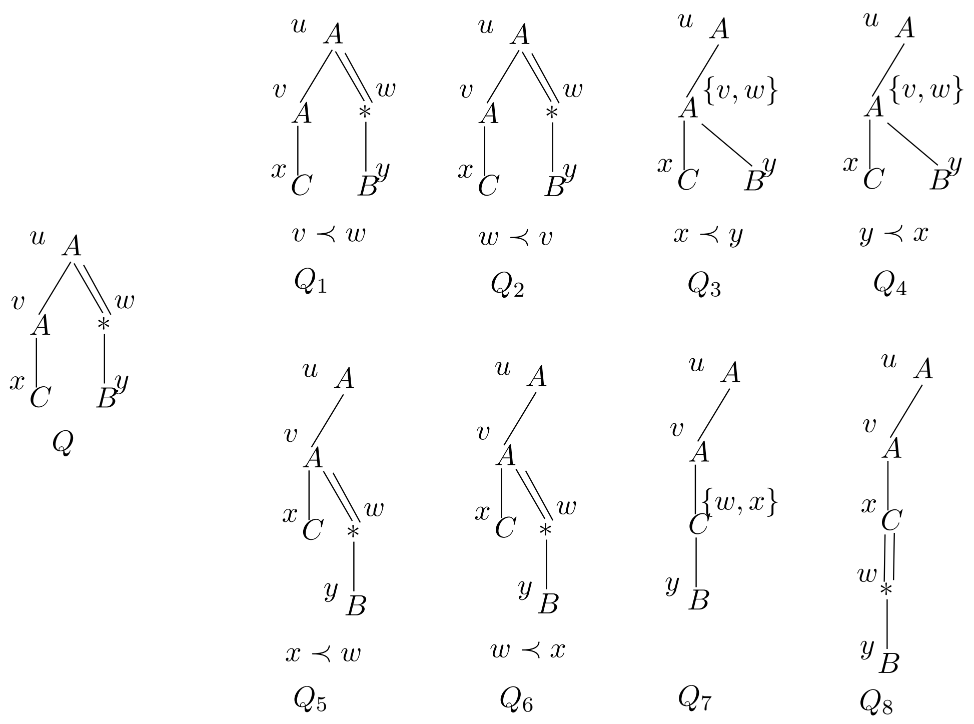

Figure 3: Query Q and complete expansion $Exp(Q) = \{Q_1, \ldots, Q_8\}$.

4. removing edges (u, v) and (u, w) from $E_{/\!/}$;
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, $Exp_{\{v,w\}}(Q)$ 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 2. The multiset $Exp_{\{v,w\}}(Q)$ contains exactly the queries $Q_1, \ldots, Q_7$. Note that if Q were of a different form, then $Exp_{\{v,w\}}(Q)$ might contain less queries. For example, if the incoming edge of v was not a descendant edge, then $Exp_{\{v,w\}}(Q)$ would not contain Q_4, Q_6 or Q_7. □

Remark 3.2. In Figure 2, 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* $Q_1 = (V_1, E_{/_1} \cup E_{/\!/_1}, r_1, \prec_1, \lambda_1, \bar{o}_1)$ *and* $Q_2 = (V_2, E_{/_2} \cup E_{/\!/_2}, r_2, \prec_2, \lambda_2, \bar{o}_2)$ *are* isomorphic, *denoted* $Q_1 \sim Q_2$, *if there exists a bijective mapping* φ *from* V_1 *to* V_2 *such that*

- $(v, w) \in E_{/_1}$ *if and only if* $(\varphi v, \varphi w) \in E_{/_2}$;
- $(v, w) \in E_{/\!/_1}$ *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, $\lambda_1(v) = \lambda_2(\varphi v))$;
- $\varphi \bar{o}_1 = \bar{o}_2$.

For example, in Figure 2, 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 order-unknown sibling nodes. Then,* $Q \equiv Exp_{\{v,w\}}(Q)$.

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 order-unknown sibling nodes, or *llou sibling nodes* for short, if u, v are order-unknown sibling nodes and there is no pair of order-unknown sibling nodes with a lower level (i.e., closer to the root) than u, v. Note that there may be several lowest-level pairs of order-unknown sibling nodes. A *complete expansion* of a multiset of queries $\mathcal{Q}$, denoted $Exp(\mathcal{Q})$, is derived by the following process:

Computing a Complete Expansion

While $\exists Q \in \mathcal{Q}$ that is not completely ordered,
- choose a pair of llou sibling nodes v, w, and
- replace Q with $Exp_{\{v,w\}}(Q)$ in $\mathcal{Q}$.

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.*

We now conclude the main result of this section.

COROLLARY 3.6. *Let $\mathcal{Q}$ be a multiset of queries. Then, $Exp(\mathcal{Q})$ is a finite completely ordered multiset of queries, such that $\mathcal{Q} \equiv Exp(\mathcal{Q})$.*

EXAMPLE 3.7. Consider first query Q in Figure 2. All queries in $\{Q_1, \ldots, Q_7\}$ are completely ordered, and thus, $Exp(Q) = \{Q_1, \ldots, Q_7\}$. Note that a complete expansion was derived by a single step of the expanding process.

A more sophisticated example appears in Figure 3. Consider the query Q in this figure. It is possible to show that $Exp(Q) = \{Q_1, \ldots, Q_8\}$. Note that these queries are derived by repeatedly choosing pairs of order-unknown 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 $Q \equiv \{Q_1, \ldots, Q_8\}$. □

4. 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).

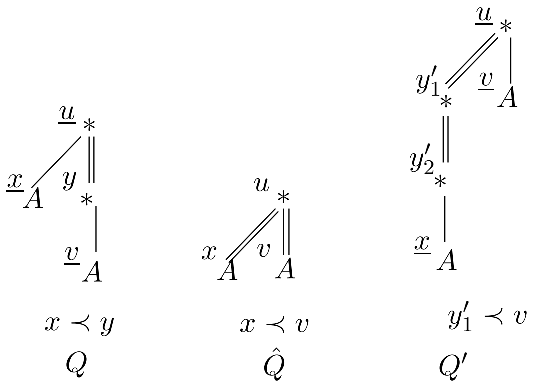

Figure 4: Query Q, and its core $\hat{Q}$. The core of Q' is isomorphic to $\hat{Q}$. In Q and Q', core nodes are underlined.

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. *$\lambda(v) \neq *$;*
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.

Observe that a node v is *not a core node* only if v is an intermediate node of Q (not the root or leaf), is labeled with a wildcard, has a single child and is not an output node.

Consider a path in Q of the form $p = u, z_1, \ldots, z_n, v$ where $u, v \in \hat{V}$, and $z_i \notin \hat{V}$, 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 $*$. The notion of a core path will be central to many of the results in this section.

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

- removing all nodes $z_1, \ldots, z_n$;
- adding a descendant edge (u, v);[3]

[3]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.

- replacing any occurrence of z_1 in $\prec$ by v (i.e., v inherits all sibling orderings of z_1).

Obviously, the query $Q_{\Downarrow(u,v)}$ 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 4. 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 φ 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 φv 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 4. 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 Q, as $\hat{Q} \sim \hat{Q}'$. Using this notational convention, $\hat{Q}$ and $\hat{Q}'$ are actually identical (and are exactly the query in the center of Figure 4).

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 $\mathcal{Q}$ and $\mathcal{Q}'$ are *isomorphic*, written $\mathcal{Q} \sim \mathcal{Q}'$ if (1) $|\mathcal{Q}| = |\mathcal{Q}'|$ and (2) there is a bijection π from $\mathcal{Q}$ to $\mathcal{Q}'$ such that

$$\forall Q \in \mathcal{Q}(Q \sim \pi(Q))\,.$$

Obviously, if $\mathcal{Q} \sim \mathcal{Q}'$, then $\mathcal{Q} \equiv \mathcal{Q}'$. 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 $(z_{i-1}, z_i) \in E_{/\!/}$ and $(z_i, z_{i+1}) \in E_/$ in Q, then $(z_{i-1}, z_i) \in E'_/$ and $(z_i, z_{i+1}) \in E'_{/\!/}$ in Q').

To demonstrate, consider queries Q and Q' from Figure 5. It is easy to see that Q' is the y-flip of Q. Next we will show that this implies that $Q \equiv Q'$.

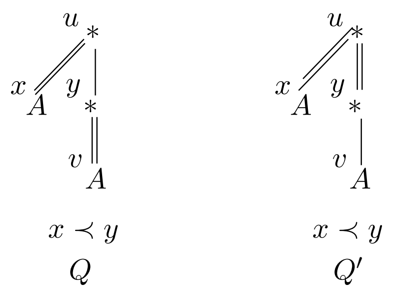

Figure 5: Queries Q and Q', such that Q' is the y-flip of Q.

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, $\hat{Q} \sim \hat{Q}'$ and $Q \equiv Q'$.*

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'.[4]

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., $\mathcal{Q} \sim^f \mathcal{Q}'$ if there is a bijection π from $\mathcal{Q}$ to $\mathcal{Q}'$ such that for all $Q \in \mathcal{Q}$, we have $Q \sim^f \pi(Q)$.

Corollary 4.3 follows from Proposition 4.2.

Corollary 4.3. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be multisets of queries. If $\mathcal{Q} \sim^f \mathcal{Q}'$, then $\mathcal{Q} \equiv \mathcal{Q}'$.*

4.2.2 Edge Unrolling

We now consider a second case where non-isomorphic queries can be equivalent. Let $Q = (V, E_/ \cup E_{/\!/}, r, \prec, \lambda, \bar{o})$ be a query. Let (u,v) be an edge in $E_{/\!/}$. 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 $E_{/\!/}$ and added to $E_/$.
- Q_2 is the query derived by (1) adding a node z with $\lambda(z) = *$, (2) replacing all occurrences of v in $\prec$ with z and (3) removing edge (u,v) from $E_{/\!/}$ and adding edges (u,z) to $E_/$ and (z,v) to $E_{/\!/}$. 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 $\{Q_1, Q_2\}$.

[4]Note that we are following the convention stated earlier that uses the same node names to denote corresponding nodes in queries with isomorphic cores.

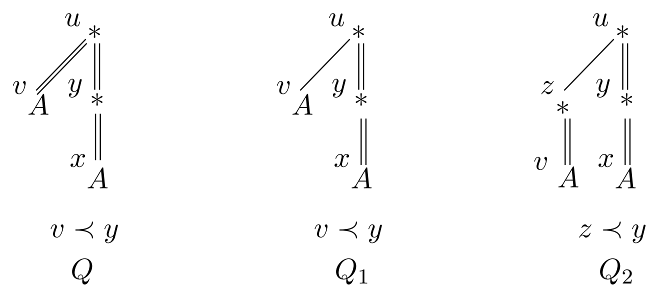

Figure 6: Queries $\{Q_1, Q_2\}$ are the (u, v)-unrolling of Q.

Example 4.4. To demonstrate, observe that $\{Q_1, Q_2\}$ in Figure 6 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 μ for which $\mu(v)$ is a child of $\mu(u)$ and Q_2 captures matchings μ' 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, 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 $\mathcal{Q}$ is k-unrolled, if every query in $\mathcal{Q}$ is k-unrolled. Given a multiset of queries $\mathcal{Q}$, the definition of a (u, v)-unrolling of a query immediately provides us with a method to create a multiset $\mathit{Unroll}_k(\mathcal{Q})$ that is k-unrolled and is equivalent to $\mathcal{Q}$. Actually, there may be many different ways to create k-unrolled multiset of queries equivalent to Q, as there may be several descendant edges on the path from u to v (which can be unrolled). To make $\mathit{Unroll}_k(\mathcal{Q})$ unambiguous, we will assume that we always unroll the topmost descendant edge on the path.

We will write $\mathcal{Q} \sim_k^f \mathcal{Q}'$ if the k-unrollings of $\mathcal{Q}$ and $\mathcal{Q}'$ are flip isomorphic, i.e., if $\mathit{Unroll}_k(\mathcal{Q}) \sim^f \mathit{Unroll}_k(\mathcal{Q}')$. Later, in Theorem 4.10, *we will prove that given multisets of completely ordered queries $\mathcal{Q}$ and $\mathcal{Q}'$ it is possible to determine a value k such that $\mathcal{Q} \equiv \mathcal{Q}'$ if and only if $\mathcal{Q} \sim_k^f \mathcal{Q}'$.* To show this, we will need to consider databases of a specific form, called canonical databases, which are introduced in the next section.

Example 4.6. In Figure 6, 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 $Q \sim_k^f \{Q_1, Q_2\}$ 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'_i are new nodes labeled $*$. 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 θ is a function that maps each pair of core endpoints in Q to a non-negative integer, then the *θ-path replacement* of Q, denoted Q_θ 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.[5] The *wildcard eliminated* version of Q, denoted $Q_{*\Rightarrow Z}$, is derived by replacing all $*$ 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.

Finally, let θ be a mapping of all pairs of core endpoints in Q to nonnegative integers. The *canonical database* for θ and Q, denoted D_θ^Q is defined as

$$D_\theta^Q := ((Q_\theta)_{*\Rightarrow Z})_{/\!/\Rightarrow /} .$$

In other words, to derive a canonical database for θ 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_θ.

Example 4.7. Consider the queries Q, its core $\hat{Q}$ and query Q', appearing in Figure 4. Figure 7 contains two canonical databases for Q, namely D_{θ_1} and D_{θ_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_{θ_1}, but is satisfiable over D_{θ_2}. □

[5]Since Σ is infinite, we may assume that there is some label Z not appearing in any queries.

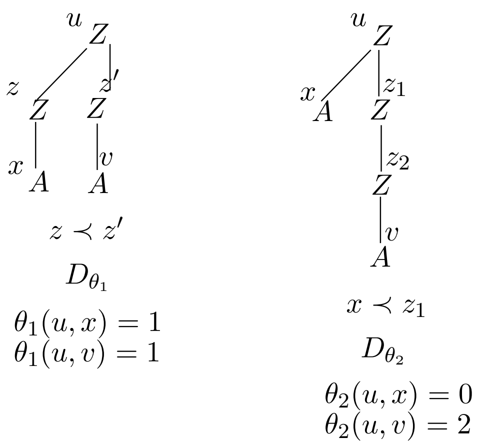

Figure 7: Canonical databases.

We continue with our notational abuse by using the same letters to denote core nodes and their corresponding database nodes (as in Figure 7). 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_θ^Q.

We now establish which types of queries may return the target series when evaluated over D_θ^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 $*$.

PROPOSITION 4.8. *Let D_θ^Q be a canonical database for θ 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:*

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

We say that θ is a *path respecting* mapping of core paths in Q to nonnegative integers, if for all pairs of core endpoints u, v in Q,

- if the core path $p = u, \ldots, v$ contains only child edges, then $\theta(p)$ is the number of intermediate nodes of p;
- if the core path $p = u, \ldots, v$ contains at least one descendant edge, then $\theta(p)$ is greater-or-equal-to the number of intermediate nodes of p.

This is an important property, since if θ is not path respecting for Q, then $Q(D_\theta^Q)$ will not contain the target series.

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

Let Q be a query and let θ 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

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

PROPOSITION 4.9. *Let Q be a query and θ 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_θ^Q with multiplicity*

$$\Phi(Q, \theta) = \prod_{(u,v) \in E} b_{u,v} \,. \tag{1}$$

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 4. 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 paper. 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 $\mathcal{Q}$ and $\mathcal{Q}'$ do not satisfy a certain property, then the number of times that $\mathcal{Q}$ and $\mathcal{Q}'$ return the target series over the canonical databases are different polynomials. This proof is somewhat in the spirit of the proofs used to characterize equivalence of conjunctive queries with or without comparisons [6] but is significantly more intricate due to the presence of recursion, implied by the descendant edges.

THEOREM 4.10. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be multisets of completely ordered queries. Let $k = max_{cp}(\mathcal{Q} \cup \mathcal{Q}') + 1$. Then,*

$$\mathcal{Q} \equiv \mathcal{Q}' \iff \mathcal{Q} \sim_k^f \mathcal{Q}' \,.$$

The intuition follows. It follows from Propositions 4.2 and 4.5 that if $\mathcal{Q} \sim_k^f \mathcal{Q}'$, then also $\mathcal{Q} \equiv \mathcal{Q}'$. Thus, it remains to show the other direction. In fact, we show the contrapositive, i.e., that if $\mathcal{Q} \not\sim_k^f \mathcal{Q}'$, then also $\mathcal{Q} \not\equiv \mathcal{Q}'$.

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

Among the queries in $\mathcal{Q}$ and in $\mathcal{Q}'$, 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. We will be creating a family of canonical databases for this query and will show that over some database in this family $\mathcal{Q}$ and $\mathcal{Q}'$ return the target series a different number of times.

By Proposition 4.8 and our choice of Q_1, it follows that any query in $\mathcal{Q}$ or $\mathcal{Q}'$ 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 $\mathcal{Q}$ and $\mathcal{Q}'$ 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 path respecting for some query in $\mathcal{Q}$ or $\mathcal{Q}'$. To choose this query, we find among all those (core-isomorphic) queries of $\mathcal{Q}$ and $\mathcal{Q}'$, a query Q_2 for which the following property holds: There is no $Q \in \mathcal{Q} \cup \mathcal{Q}'$, such that for all pairs of core endpoints u, v, the core path from u to v in Q is shorter than the core path from u to v in Q_2.

Now, our family of databases is defined to be canonical databases $D_\theta^{Q_2}$ such that θ is path-respecting for Q_2. Finally, we show that given such a canonical database $D_\theta^{Q_2}$, the number of times that $\mathcal{Q}$ and $\mathcal{Q}'$ return the target series is a polynomial in the values assigned by θ. Using Proposition 4.9 we show that the polynomials for $\mathcal{Q}$ and $\mathcal{Q}'$ differ, implying that there is some canonical database for which $\mathcal{Q}$ and $\mathcal{Q}'$ return the target series a different number of times.

Taken together, Theorem 4.10 and Corollary 3.6 provide an equivalence characterization for arbitrary multisets of queries (that may not be completely ordered).

Corollary 4.11. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be multisets of queries, which may not be completely ordered. Then,*

$$\mathcal{Q} \equiv \mathcal{Q}' \iff Exp(\mathcal{Q}) \sim_k^f Exp(\mathcal{Q}')\,,$$

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

Based on Corollary 4.11, we present an upper bound on the complexity of equivalence. The following result is relies on the facts that (1) the size of every query in the multiset $Unroll_k(Exp(\mathcal{Q}))$, $Unroll_k(Exp(\mathcal{Q}'))$ 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 $Unroll_k(Exp(\mathcal{Q}))$, $Unroll_k(Exp(\mathcal{Q}'))$ at the same time.

Theorem 4.12. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be multisets of queries, which may not be completely ordered. It is possible to determine whether $\mathcal{Q} \equiv \mathcal{Q}'$ in PSPACE.*

We now consider the containment problem. As is the case for Datalog queries, containment is significantly more difficult than equivalence. In fact, even when $\mathcal{Q}$ and $\mathcal{Q}'$ are orderless, determining whether $\mathcal{Q} \subseteq \mathcal{Q}'$ is undecidable. The proof of Theorem 4.13 is in the spirit of a similar result for bag containment of unions of Datalog queries in [12].

Theorem 4.13. *The problem of deciding bag containment among orderless multisets of queries is undecidable.*

5. HORIZONTAL AXES

Until now, we considered queries which use the $\prec$ relationship among sibling nodes as a horizontal axis. As noted earlier, $\prec$ 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 2.1, we will call the latter *standard queries.*

5.1 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). We say that b is an *immediately-following sibling* of a if b is a following sibling of a, and there is no c such that $a \prec_D c$ and $c \prec_D b$. 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 $c \prec b$. The relationship last-child is defined analogously.

5.2 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 μ of nodes in Q to those in a database D is a *matching* of Q to D if it satisfies all conditions in Definition 2.2, and moreover,

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

Remark 5.1. 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 pattern to nodes with no apparent parent, such as c. The following relationship is then expressed using $\prec_f$, and not with the tree structure. $\square$

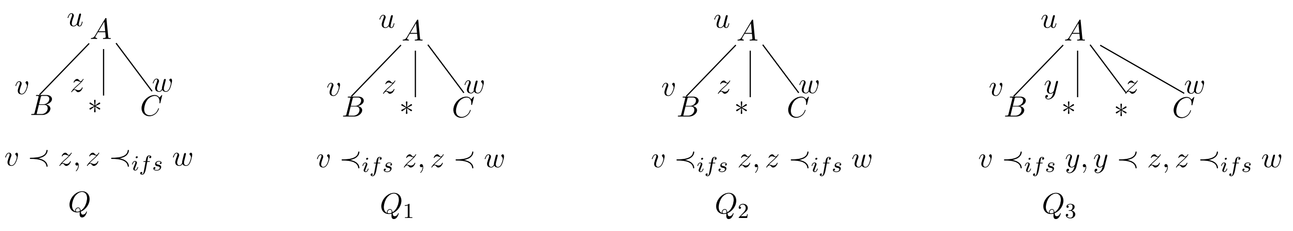

Figure 8: Queries using the immediately-following sibling axis.

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

Basically, the idea behind Proposition 5.2 is to start by computing $Exp(Q)$, in the manner described earlier, while ignoring the new relations. Then, for all v, w such that $v \prec_f w$, we eliminate queries in which w does not follow v. Similarly, for all v, w such that $v \prec_{fs} w$, we eliminate queries in which w is not a following sibling of v. Finally, in the remaining queries, for all $v \prec_{fs} w$, we replace any descendant incoming edges to v and w with child edges.[6] At this point all relationships in $\prec_f$ and $\prec_{fs}$ can be dropped. Care must be taken in the actual reduction as the nodes v from Q may not appear directly in queries of $Q' \in Exp(Q)$ due to node merges. Therefore, the required relationships are actually checked not between v and w, but rather between corresponding nodes in Q'.

We immediately derive the following corollary.

COROLLARY 5.3. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ 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 $\mathcal{Q} \equiv \mathcal{Q}'$ in PSPACE.*

5.3 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 children relationship.

Once again, to define query semantics, we extend the notion of a matching in the natural fashion, i.e., a mapping μ of nodes in Q to those in a database D is a *matching* of Q to D if it satisfies all conditions in Definition 2.2, and moreover,

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

[6]Such a replacement is correct due to the semantics of $\prec$.

Unlike following and following sibling, the first axis does increase the expressive power of our query language, and thus, we cannot show a result similar to Proposition 5.2. Instead, we show the following result.

PROPOSITION 5.4. *Let Q_1 and Q_2 be queries that may use the first axis. Then, there are multisets of standard queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ (i.e., that do not use the first axis) such that $Q_1 \equiv Q_2$ if and only if $\mathcal{Q}_1 \equiv \mathcal{Q}_2$.*

The basic idea behind Proposition 5.4 follows. We start by setting $\mathcal{Q}_1 = \{Q_1\}$. Then, while there is a query $Q \in \mathcal{Q}_1$ 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 $\mathcal{Q}_1$ and add the following queries to $\mathcal{Q}_1$

- We create a new node z, *(1)* add z as a child to u, *(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.

Intuitively, the first query captures matchings where none of the children of u is its first child in the databases, and the remaining queries choose one of the nodes of u to be its first child. When the process terminates we drop all sets F. Finally, by defining $\mathcal{Q}_2$ similarly, we get the result of Proposition 5.4. Corollary 5.5 follows.

COROLLARY 5.5. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ 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 $\mathcal{Q} \equiv \mathcal{Q}'$ in PSPACE.*

5.4 Immediately-Following Sibling Axis

We now consider queries which may use the immediately-following sibling axis. Note that XPath does not have this axis built-in, however, it is useful to express positional constraints among nodes.[7] Our queries are now of the form $Q = (V, E, r, \prec, \prec_{ifs}, \lambda, \bar{o})$, where $\prec_{ifs}$ is a partial order

[7]In fact, the immediately-following sibling axis can be used to express numerical positional constraints (such as that v is the i-th child of u), but a full discussion is beyond the scope of this paper.

corresponding to the immediately-following sibling relationship. As before, we extend the notion of a matching μ, so that it must satisfy the conditions in Definition 2.2, and in addition,

- For any pair of nodes v, w such that $v \prec_{ifs} w$ it holds that μw is the immediate following sibling of μv.

The relationship $\prec_{ifs}$ is considerably more challenging than those considered earlier. In fact, $\prec_{ifs}$ and $\prec$ interplay in a manner similar to the child and descendant axis. Thus, the immediately-following-sibling relationship can introduce new ways in which syntactically different queries may be equivalent. We demonstrate this in the following example.

Example 5.6. Consider the query Q in Figure 8. Note that z must follow v (according to the semantics of $\prec$), but w must be an immediately-following sibling of z. Basically, the node z is used to constrain there to be at least one node between v and w. Query Q_1 is equivalence to Q, although order of the relationships $\prec$ and $\prec_{ifs}$ has been *flipped*. This equivalence is similar in spirit to the flipping of edge types, introduced in Section 4.2, and demonstrated in Figure 5.

It is also possible to show that $Q \equiv \{Q_2, Q_3\}$. Intuitively, Q_2 requires there to be no nodes between v and z, while Q_3 requires there to be at least one node between v and z. Thus, $\{Q_2, Q_3\}$ can be viewed as an *unrolling* of the horizontal relationship between v and z. Such unrollings bear resemblance to the edge unrollings of Section 4.2, demonstrated in Figure 6. $\square$

It is possible to characterize equivalence between queries which use the immediately-following sibling axis, by extending two key notions introduced earlier:

- *Flip isomorphisms* should be extended to allow "flips" in the horizontal relationships, and not only in the vertical child/descendant relationships. Note that as before, flips must occur only around nodes of a special type. In this case, only $\prec, \prec_{ifs}$ around leaf nodes with the label $*$ may be flipped.
- *Unrollings* should be extended to allow pairs of adjacent nodes with the $\prec$ relationship to be "unrolled" by adding new nodes in between. These new nodes will be leaves, labeled $*$. Care must be taken to avoid increasing multiplicities by adding the new nodes. This is achieved by using the $\prec_{ifs}$ relationship, which implies that a matching of a node is functionally dependent on the mapping of its previous sibling.

A characterization similar to that of Theorem 4.10 can be shown, if the horizontal unrollings are of length greater than the largest branching degree of the queries. The full proof is rather intricate, requires an adaptation of the notion of query cores (which ignores nodes that are leaves, labeled $*$), an adaptation of the notion of canonical databases (which can extend core queries horizontally, and not only vertically) and finally, careful examination of the polynomials which characterize result multiplicity. A complete discussion is beyond the scope of this paper.

6. CONCLUSION

In this paper we characterized bag equivalence of XPath queries. Our results are general, and allow descendants, wild-cards, branching, unions, multiple output nodes and horizontal axes. This paper is 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 XPath queries, 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.

Acknowledgments

This research is partially supported by the Israel Science Foundation (Grant 143/09).

7. REFERENCES

[1] A. Berglund. Extensible stylesheet language (XSL) version 1.1. http://www.w3.org/TR/xsl, 2006.

[2] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML query language. http://www.w3.org/TR/xquery, 2007.

[3] S. Chaudhuri and M. Y. Vardi. Optimization of *real* conjunctive queries. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC*, pages 59–70. ACM Press, 1993.

[4] J. Clark and S. DeRose. XML path language (XPath) version 1.0. http://www.w3.org/TR/xpath, 1999.

[5] S. Cohen. Equivalence of queries that are sensitive to multiplicities. *VLDB J.*, 18(3):765–785, 2009.

[6] S. Cohen, W. Nutt, and Y. Sagiv. Deciding equivalences among conjunctive aggregate queries. *J. ACM*, 54(2), 2007.

[7] S. Cohen, Y. Sagiv, and W. Nutt. Equivalences among aggregate queries with negation. *ACM Trans. Comput. Log.*, 6(2):328–360, 2005.

[8] M. Davis. *Computability and Unsolvability.* Dover Publications, 1982.

[9] S. DeRose, E. Maler, and R. Daniel, Jr. XML pointer language (XPointer) version 1.0. http://www.w3.org/TR/WD-xptr, 2001.

[10] S. DeRose, E. Maler, and D. Orchard. XML linking language (XLink) version 1.0. http://www.w3.org/TR/xlink, 2001.

[11] A. Deutsch and V. Tannen. Containment and integrity constraints for xpath. In *KRDB*, Rome, Italy, Sept. 2001.

[12] Y. E. Ioannidis and R. Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *ACM Trans. Database Syst.*, 20(3):288–324, 1995.

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

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

[15] B. ten Cate and C. Lutz. The complexity of query containment in expressive fragments of xpath 2.0. In *PODS*, pages 73–82, Beijing, China, June 2007.

[16] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1: Structures. `http://www.w3.org/TR/xmlschema-1`, 2004.

[17] P. T. Wood. Containment for xpath fragments under dtd constraints. In *ICDT*, pages 297–311, Siena, Italy, Jan. 2003.