Algebraic Constructs for Querying Provenance

Murali Mani, Mohamad Alawa, Arunlal Kalayanasundaram
University of Michigan, Flint
303 E. Kearsley St, Flint, MI 48502
{mmani, malawa, arunlalk}@umflint.edu

Abstract— **Provenance that records the derivation history of data is useful for a wide variety of applications, including those where an audit trail needs to be provided, where the trust-level attributed to the sources contribute to determining the trust-level in results etc. There have been different efforts for representing provenance information, the most notable being the Open Provenance Model (OPM). OPM defines structures for representing the provenance information as a graph with nodes and edges, and also specifies inference queries that can be expressed in Datalog/SQL. However, the requirements of a query language for provenance information go much beyond those that can be expressed using only inference queries. In our work, we build on OPM and propose two classes of algebraic constructs for querying provenance information: content-based operators that operate on the content of nodes and edges, and structure-based operators that operate on the graph structure of the provenance graph. An user can express a query as a workflow by composing these content-based and structure-based operators. Our operators are powerful, and an user can express a wide variety of interesting queries on the provenance data, that go much beyond simple inference queries as expressible using Datalog/SQL. As part of our evaluation, we show different queries and how they can be expressed using our constructs.**

Keywords-Provenance; graph; data model; query language; algebraic operators

I. INTRODUCTION

Provenance (also referred to as lineage, parentage, pedigree, genealogy, or filiation) describes the steps by which data was derived. Provenance has been found useful for a wide variety of scenarios, where one needs to determine the attribution of a data item. For instance, a doctor may need to find the sources from which a data item was obtained to determine whether the item is clinically valid; in data gathering and analysis, the sources from which individual data items are obtained is used for later verification.

In [6], the authors describe two different granularities of provenance. In workflow provenance, which is coarse-grained, each process in the workflow manipulates a set of data items producing another set of data items; the granularity for provenance recording and querying is at the level of data items. In data provenance, which is fine-grained, the queries deal with identifying which pieces of a data item contributed to form a piece of another data item. A well-studied example of data provenance is in the context of SQL queries, where users may want to find why a particular row is in the result (or in some cases, why it is not in the result) [4, 7, 9, 10, 13, 18, 24]. Our focus in this work is on workflow provenance as studied in [1, 5, 15, 16, 22]. In [12] as well as in [14], the authors describe a single system that supports both data and workflow provenance.

Initial works on provenance were tightly integrated with the application they dealt with, such as scientific workflows [1, 5, 8, 15]. OPM provenance model [20] was developed to serve as an application-independent model for representing provenance. OPM represents the provenance information of an application as a graph, where nodes represent artifacts (data items), processes (that manipulate data items), and agents (that control processes), and edges represent causal dependencies. A simple fictitious example of milk powder production, distribution and export is shown in Fig. 1. It shows collection of milk, production of milk powder, distribution at restaurants and at retail stores, exporting, and candy manufacturing and distribution. As in OPM, an ellipse is used to represent an artifact, and a rectangle is used to represent processes. The causal dependencies are shown in a simplified manner than OPM for easier understanding.

While OPM explains how provenance information can be represented, the manipulation of provenance information is limited to inference queries, such as determine all the items that are directly or indirectly produced by a particular data item. An example based on the example application shown in Fig. 1 is to determine the candy bars that are affected by a particular set of milk powder. These queries can be represented in Datalog or using SQL [17].

However, limiting ourselves to such inference queries as provided by OPM is insufficient for a variety of applications, and several interesting queries cannot be expressed. Two example queries that cannot be expressed using inference queries as provided by OPM are described below.

Query1 (shortest path): Suppose there is a time duration associated with processes in Fig. 1, that shows the time it takes for the process from start to finish. If a particular set of milk powder is found to be contaminated, we may want to find the earliest set of products in domestic market that are produced by this contaminated set of milk powder. This requires shortest path type of queries.

Query 2 (sub-graph matching): Suppose there is a pattern that the user is looking for, which can be specified as a graph (and using regular expressions to describe paths). A user may want to find the occurrences of this pattern in the provenance graph. An example with regards to Fig. 1 is finding cases of milk powder produced in Michigan being used for candy manufacturing in France and sold in French retail stores.

There is considerable work in the graph database community on graph query languages and graph algebras, such as [11]. However, these are too generic and often not practical for provenance data. Furthermore, these languages focus mainly on sub-graph isomorphism, and cannot express

 ISBN: 978-1-61208-185-4

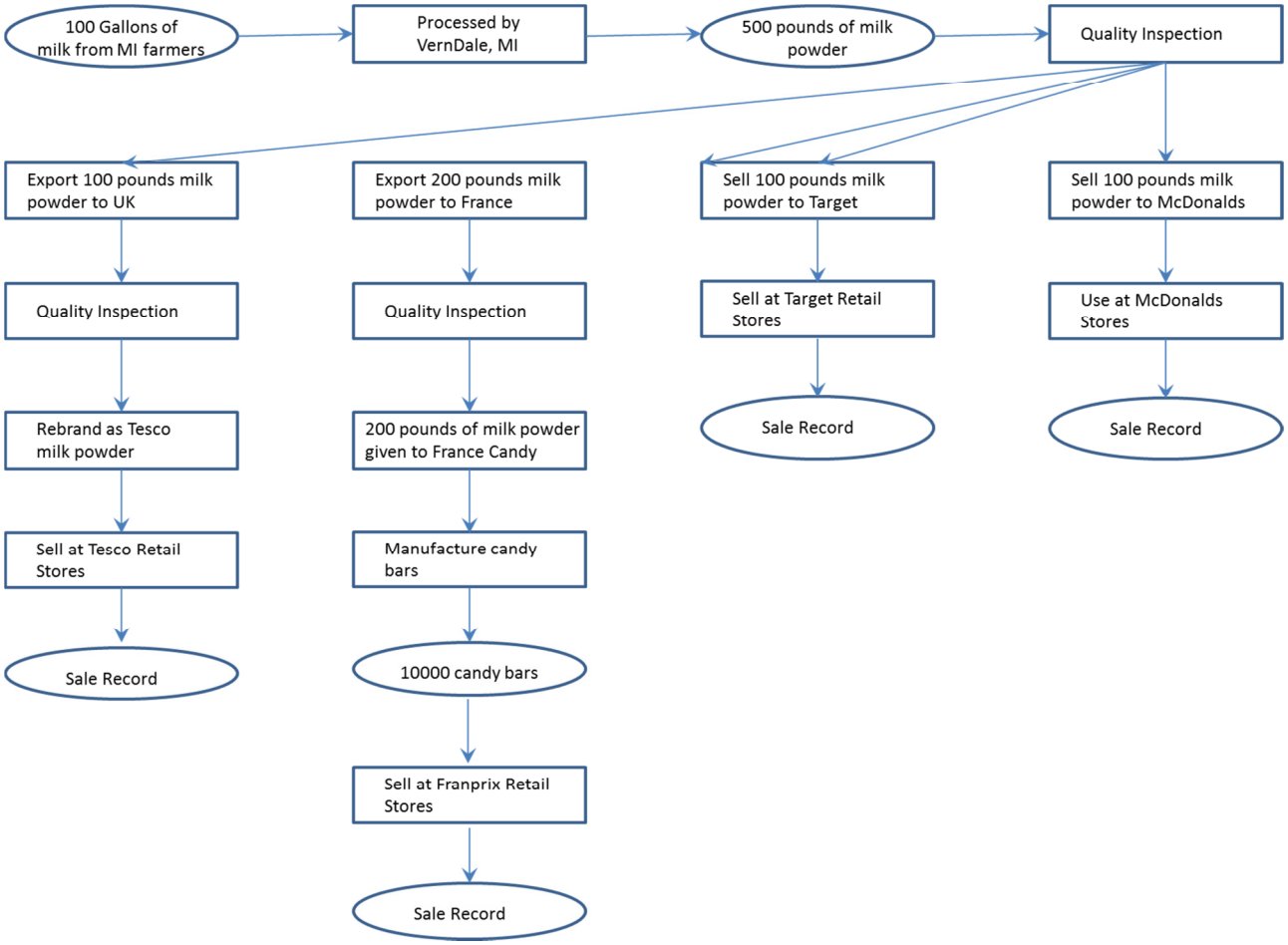

Figure 1. An example application illustrating provenance data. Of the 500 pounds of milk powder produced, 200 pounds are used for local sales, and 300 pounds are exported. Of this, 200 pounds exported to France are used to manufacture candy bars, which are then sold at retail stores.

queries such as shortest path. Recently, OPQL [16], a query language for provenance data was developed to support user friendly languages for manipulating provenance; their query language is motivated by [11]. An OPQL query is translated into an SQL query for processing; this implies that OPQL cannot express wide range of queries such as shortest path queries, which cannot be expressed in Datalog/SQL.

In this paper, we propose generic algebraic operators for querying provenance. We classify our operators into two categories: content based operators that query the content information, and structure based operators that query the graph structure of provenance. A user can express a query as a workflow that integrates these two categories of operators seamlessly. This provides a powerful framework that can be used for answering a wide variety of useful provenance queries, including shortest path and sub-graph matching.

Our contributions in this paper are as follows:

1) We define a set of algebraic operators for querying the content of nodes and edges in provenance graph, as well as the structure of the provenance graph. Our operators are powerful, yet simple, and we believe they will form the basis for any future provenance query languages.
2) A user can express a query as a workflow that integrates both content-based and structure-based operators. This provides a powerful framework that can express a wide variety of useful provenance queries, much beyond what can be expressed using Datalog/SQL based languages.
3) We evaluate the expressiveness of our query model. We consider our own example application (Fig. 1), and show how different queries can be expressed. We also consider some queries from the third provenance challenge [23], and study how these queries can be expressed using our query model.

Outline: The outline for the rest of the paper is as follows. In Section 2, we outline the OPM data model [20] that we use for representing the provenance data; we simplify the OPM data model so that readers who are not aware of the OPM data model are not carried away by the details, and can focus on the query model. In Section 3, we describe the algebraic constructs that we define. We divide this section into three parts; we first describe the content based constructs, we then describe the structure graph based constructs, finally we describe how a user can express a query by integrating these constructs. In Section 4, we describe our evaluation, where we examine different queries and how they can be expressed in our model. Section 5 discusses related work, and Section 6 concludes this paper.

 ISBN: 978-1-61208-185-4

II. REPRESENTATION OF PROVENANCE INFORMATION

Let us first examine in some detail as to how the provenance information is represented. Graph based data models have been used for provenance in the past, including for OPM [20], and also other works such as [13]. OPM describes three kinds of nodes: *artifacts*, which represent a state of an object at a particular time instant and is immutable; *processes*, which represent an action or a series of actions performed on artifacts and producing new artifacts; *agents*, which represent contextual entities that act as a catalyst for a process. In OPM, an artifact is represented as an ellipse, a process as a rectangle and an agent as an octagon. In our running example in Fig. 1, the 500 pounds of milk powder is an artifact, quality inspection is a process. Any artifact, process or agent may have annotations, which are additional information associated with these entities. The annotations are represented as a set of property-value pairs. Example annotations that can be added to the example shown in Fig. 1 include: range of serial numbers of the milk powder cans produced; time it takes for the process that processes milk and produces milk powder.

OPM also describes five kinds of edges (edges are called dependencies in OPM): *used* representing processes that used an artifact; *wasGeneratedBy* representing artifacts that were generated by a process; *wasControlledBy* representing processes that were controlled by an agent; *wasDerivedFrom* representing artifacts that were derived from an artifact (used for a dataflow view of provenance); *wasTriggeredBy* representing processes that were trigged by a process (used for a process oriented view of provenance). In Fig. 1, the process "Processed by Verndale" *used* 100 gallons of milk; 500 pounds of milk powder *wasGeneratedBy* the process "Processed by Verndale". The process "Sell at Tesco Retail Stores" *wasTriggeredBy* the process "Rebrand as Tesco Milk Powder". In our data model, we do not explicitly indicate the type of the edge; the type of an edge can be immediately inferred based on the nodes that the edge connects.

OPM also describes multi-step inferences and multi-step edges, which are based on transitive closure kind of operators (widely used in the context of directed graphs). For instance, the process "Sell at Tesco Retail Stores" has a multi-step *wasTriggeredBy* dependency on the process "Export milk powder to UK". Also two automatic completion rules are described in OPM.

Our work focuses on query language constructs for provenance. In this paper, we continue to use the same types of nodes and edges as described in OPM. However, we simplify the model and do not distinguish between the different types of edges (as mentioned above, the type of an edge can be inferred based on the types of nodes that the edge connects).

III. ALGEBRAIC CONSTRUCTS

In this section, we define our query language constructs. We divide our query language constructs into two categories: content based constructs that work on the annotations associated with the nodes and edges, and structure based constructs that work on the graph structure of the provenance graph. We will then describe how these two sets of constructs can be integrated to form a powerful query model, that can express a wide variety of interesting queries.

A. Constructs for Querying Content

The content-based query constructs are used to select nodes or edges based on the annotations associated with them. In short, these constructs operate on a set of entities (which can be nodes or edges), and produce a set of entities as a result; the graph structure of provenance such as edges or paths between two nodes is not used by these constructs.

Some of the operations that are expressible using content based query constructs are listed below. See that our content based query constructs are similar to those in relational algebra, and we borrow notations from relational algebra to represent the different query constructs.

1. Select a set of entities based on a selection condition (used to select nodes and edges based on annotations). This is expressed as $\sigma(S, c)$, where S is a set of entities, and c is the condition. The result of selection is $S' \subseteq S$; each entity in S' satisfies the condition specified in c.

   Example 1 (Select nodes based on a filtering condition): In Fig. 1, select all processes that are involved with "selling at a retail store". The result of this would include: selling of milk powder at Tesco retail stores, selling of milk powder at Target retail stores, and selling of candy at Franprix retail stores.

   Suppose the process "Selling at Tesco Retail" is as follows (other processes have similar representation):
   process-id = PTESUK101
   annotation:
   process type: Sell
   description: Sell at Retail Store
   duration of process: Jan 10, 2011 – Jan 25, 2011
   details of store: Tesco Retail, Cardiff
   Then the selection condition can be represented as annotation.description LIKE "Sell at Retail Store". Note that the above is very similar to the select operator in relational algebra; we need to use path expressions similar to XPath (XML path language) as annotations can be arbitrarily nested.

2. Project properties, such as ids or annotations from a set of nodes. This is similar to the project operator in relational algebra, and is denoted as $\pi_{aList}(S)$, where S is the set of entities and *aList* is the set of properties of these entities that are to be projected. The result is a set of entities corresponding to the *aList* that is projected. Because annotations can be arbitrarily nested, we use a path expression to indicate annotations to be projected.

   Example 2 (projection of process-id): Find all the process-ids from the set of processes shown in Fig. 1.

3. Set operations such as union, intersection, difference, cross product of two sets of nodes.

4. Aggregate operations such as min, max, sum, count, average on a set of entities.

   Example 3 (aggregation): How many processes are involved with "selling at a retail store". This involves first selecting processes that are involved with "selling at a retail store" (Example 1), and then counting the number of processes that are the result of the selection.

B. Constructs for Querying Structure

The constructs for querying structure use the graph structure of the provenance graph as the basis of querying. Remember that the provenance graph consists of a set of nodes and edges, with each edge connecting two nodes (as shown in Fig. 1). Let us first examine a set of basic functions that are used to further define additional operators. The basic functions include:

1. *from(e)*: returns the node from where the edge e starts.
2. *to(e)*: returns the node where an edge e ends.
3. $from^{-1}(n)$: returns the edges that start from node n.
4. $to^{-1}(n)$: returns the edges that end in node n.
5. *next(n)*: returns all the nodes such that there is an edge from node n to it.
6. *prev(n)*: returns all the nodes, such that there is an edge from that node to n.

Example 4: In Fig. 1, let QIMich denote the node depicting the quality inspection process that took place in Michigan. The query next(QIMich) produces four nodes: export 100 pounds to UK, export 200 pounds to France, sell 100 pounds to Target, sell 100 pounds to McDonalds.

We can write quite complex queries using the above functions. For instance, to find all nodes that have a path of length 2 from QIMich, we can combine next operators; first, a next operator finds all the nodes reachable from QIMich by a path of length 1; use the next operator again to find paths of length 1 from these nodes (implying a path of length 2 from QIMich). These can be expressed in Datalog (requires a join). Assume the relation nextTable(d, s) indicates that node d is a node in next(s). The relation pathOfLen2 defines the nodes that are at a path of length two from the node QIMich.

pathOfLen2(y) :- nextTable(x, QIMich), nextTable(y, x).

In our work, instead of defining such operators, we define a very powerful general selection operator on the structure graph of provenance, that takes a structure graph as input, and produces another structure graph as output. Let *G(N, E)* denote a provenance graph with *N* as the set of nodes and *E* as the set of edges. Let f_n be a selection condition on the nodes, and f_e be a selection condition on edges. The selection operator, denoted as σ_g, is defined as:

$\sigma_g(G, f_n, f_e) = G' = (N', E')$, where

$N' \subseteq N$, is the set of nodes that satisfy the condition in f_n,

$E' \subseteq E$, is the set of edges that satisfy the condition in f_e, and the nodes connected by an edge in *E'* are both in *N'*.

See that the selection operator takes a provenance graph *G* as input, along with f_n and f_e; the result of the selection is another graph *G'*, which is a sub-graph of the original graph. We require that edges chosen must only be those such that the nodes connected by an edge must be in *N'*. The reason for this is that we need to select both the nodes connected by an edge for that edge to be selected. There are no restrictions on how to specify the conditions f_n and f_e. This allows complex conditions to be specified, much beyond what can be expressed in Datalog/SQL. At the same time, our operator maintains a lot of useful properties including the elegance of a simple algebraic operator with clear semantics. Several queries, including reachability, shortest path, sub-graph matching etc can be expressed using this general selection operator. A few example queries that can be expressed using this general selection operator are given below.

Example 5 (descendant): Given a structure graph *G = (N, E)* (graph *G* has *N* nodes and *E* edges), and a set of *N'* nodes, return the structure graph including the nodes in both *N'* and *N*, the nodes reachable in *G* starting from any node in *N'*, and include the edges between these nodes. Expressing this as a general selection operator, f_n selects the nodes:

$(N \cap N') \cup \{n \mid n \in N, \exists\, s \in N', there\ is\ a\ path\ from\ s\ to\ n\ in\ G\}$

f_e selects edges on a path in *G* starting from a node in *N'*.

For instance in Fig. 1, we can find all the descendants of the process: "Processed at Verndale, MI" if we want to determine what are all the processes and artifacts that were directly or indirectly caused by this process. The result of this selection will be the entire graph except for the first node "100 gallons of milk from MI farmers" and the edge starting from this node. One can define similarly an ancestor operator with appropriate conditions for f_n and f_e.

Example 6 (in-betweener): Given structure graph *G = (N, E)* (graph *G* has *N* nodes and *E* edges), and a set of *N'* nodes, return the structure graph including the nodes in both *N'* and *N*, all the nodes that are on a path between two such nodes, and all the edges on these paths.

This can be expressed as a general selection operator where f_n selects the nodes: $(N \cap N') \cup \{n \mid n \in N, \exists\, a, b \in N', n\ is\ a\ node\ on\ a\ path\ from\ a\ to\ b\ in\ G\}$

f_e selects edges on a path in *G* between two nodes in *N'*.

For instance in Fig. 1, we can find the graph in between the nodes "Proceessed at Verndale, MI", "Sell at Tesco Retail", and "Sell at Franprix Retail". Another example of the in-betweener operator is shown in Fig. 3.

Example 7 (path matching): For this, we first define a labeling function, that maps nodes and edges to labels. The path matching operator works on the graph after the labeling and selects all paths in the original graph that match the path pattern. For this, the selection condition f_e selects the edges:

$\{e \mid \exists p, p\ is\ a\ path\ in\ G\ after\ labeling, p\ matches\ the\ path\ pattern, and\ e\ is\ an\ edge\ in\ p\}$

f_n selects the nodes such that each edge *e* that is selected either starts from the node or ends in the node.

An example of path matching is shown in Fig. 2.

 ISBN: 978-1-61208-185-4

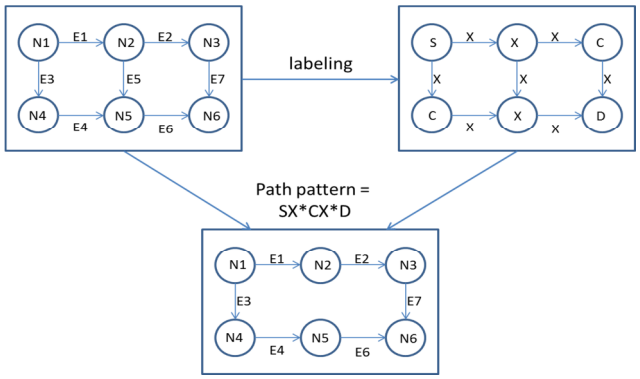

Figure 2. Illustrating path matching. Given a provenance structure graph, first the labeling is applied, and the result of matching the path pattern SX*CX*D is shown.

We call the different selection operators, each with its own definition of f_n and f_e as a variation of the general selection operator σ_g. The shortest path operator is such a variation, where f_e selects the edges that are in the shortest path given two nodes, and f_n selects the nodes that are along the shortest path between the nodes. Sub-graph isomorphism as in [11] is another variation, as the result is a sub-graph of the original provenance structure graph; another variation is an operator that returns a sub-graph pruning away nodes whose in-degree/out-degree are above/below a threshold.

We define union and intersection of provenance graphs; union of two provenance structure graphs returns a new graph that includes all the nodes in these graphs, and all the edges in these graphs; intersection of two provenance structure graphs returns a new graph that includes only the nodes present in both these graphs and the edges that are present in both these graphs.

Even though the general selection operator is powerful, there are queries that require new nodes or edges that cannot be expressed using our selection operator (because the selection operator only returns a sub-graph of the original graph). An operator, called the abstraction operator allows introducing new edges in some situations (such as transitive closure). This operator takes a provenance structure graph $G = (N, E)$ and a set N' of nodes as input, and produces a graph which includes all the nodes in $N \cap N'$. If there is an edge between 2 nodes in $N \cap N'$ in G, that edge is kept. If there is a path between 2 nodes in $N \cap N'$ in G, and no intermediate node of this path is in $N \cap N'$, then a new edge is added to the resulting structure graph (Fig. 3).

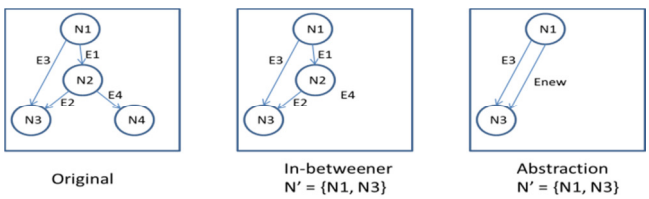

Figure 3. Example showing in-betweener operator (expressed using the general selection operator), and abstraction; both these operators consider the nodes {N1, N3} as input.

C. Integrated Query Model

In the previous two sections, we examined different operators for querying the content of provenance entities and for querying the structure of provenance graph. We need to integrate these two operator sets to be able to express a wide range of interesting queries. For this, we define one more operator that takes a structure graph and projects the set of provenance entities (nodes and edges in the graph). This projection operator is denoted as $\pi_g(G)$; it returns the set of nodes and edges in G. Different content-based constructs can be applied on this set of provenance entities. Also note that the basic structure functions (from, to, next etc) can be used to select entities from a structure graph.

In addition, any implementation of our query model will provide a pre-defined set of variations of the general selection operator. For instance, an implementation may provide descendant, ancestor and shortest-path variations of the general selection operator. Furthermore, for any operator, multiple physical level implementation alternatives can be provided.

Let us look at a fictitious, but complete example based on the provenance structure graph shown in Fig. 1. Assume that a certain batch of milk powder is found to be contaminated, and we need to find all candy bars that are affected (so that they can be recalled). Suppose we also want to find the total financial loss (the total worth of the affected candy bars).

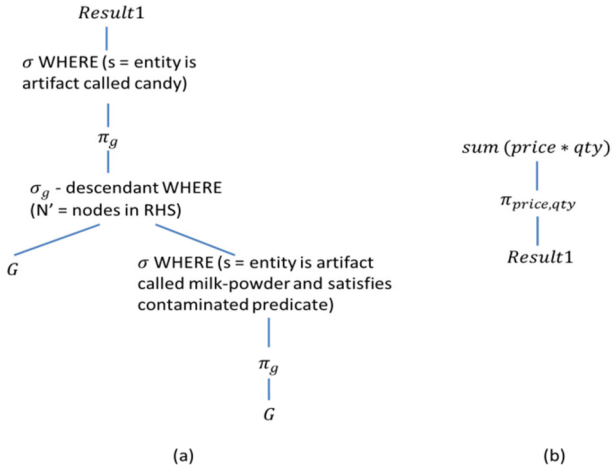

Figure 4. (a) computes the candy bars that are affected; (b) computes the total financial loss

Fig. 4 shows how this query can be answered. From the structure graph, G, the $\pi_g(G)$ operator first computes the set of entities in G; then the content-based selection operator selects the artifacts that represent the contaminated milk-powder. Now the general selection operator σ_g- descendant selects the sub-graph of G that represent descendants of the contaminated milk-powder entities. We apply $\pi_g(G)$ to get all the entities in this graph; then the content-based selection operator selects all artifacts that are candy bars. To find the total financial loss, we project (content-based projection) the price and qty of each of the selected candy bar entities from above, and perform aggregation to find the total loss.

 ISBN: 978-1-61208-185-4

IV. EVALUATION

For our evaluation, we consider two more examples from our milk powder example in Fig. 1. We will further consider some example queries from the Third Provenance Challenge [19], and describe how they can be expressed.

Query 1: From Fig. 1, determine how a problematic set of milk powder was produced, transported and processed to make an affected brand of candy.

Fig. 5 (a) shows Query 1. An alternate option (to using two content-based selection operators) is to use one selection operator with predicates combined using OR. Fig. 5 (b) shows Query 2 using shortest-path operator.

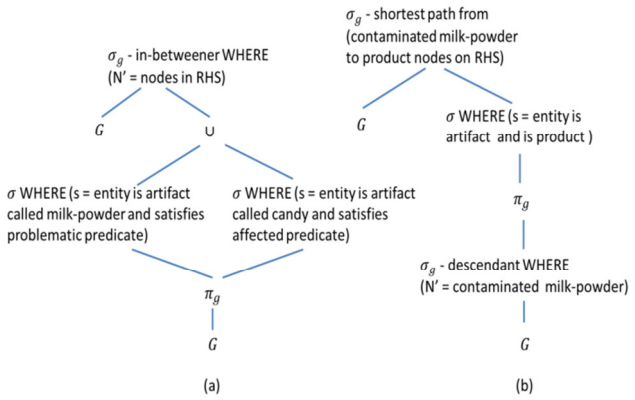

Figure 5. (a) Query 1 (b) Query 2 using shortest-path operator

Query 2: From Fig. 1, determine the earliest products produced from known contaminated milk powder.

The Third Provenance Challenge [23] provided provenance information about data from the Pan-STARRS project, that continuously scans the visible sky once a week and builds a time-series of data. The aims of the project include: help detect moving objects that may potentially impact with earth, build a massive catalog of solar system and stars. These are some of the queries from this data.

Query 3 (from [23]): Given a particular detection which files contributed to it?

This is similar to Fig. 4 (a), except that the first content-based selection selects the detection, the general selection is an ancestor operator, and the second content-based selection selects the files.

Query 4 (from [23]): The user finds a particular table with data that they do not expect. Was the range check performed on this table?

Query 4 can be answered using the path matching operator as shown in Fig. 6 (a). Fig. 6 (b) shows Query 5, also using the path matching operator.

Query 5 (from [23]): The user executes a workflow many times over different data sets. Find which of these executions halted.

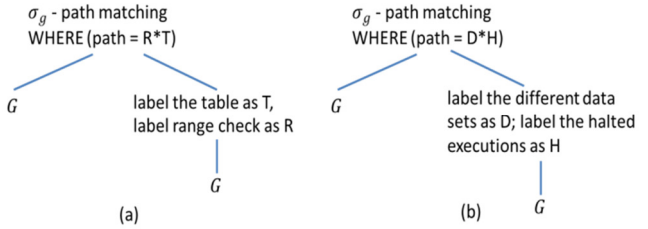

Figure 6. (a) Query 4, if there are no paths selected by the path matching general selection operator, that means range check was not performed; if there is a path selected, then range check was performed (b) Partial Query 5, showing that path matching operator can be used to find all paths resulting in a halted execution starting from the specified data sets.

From our evaluation section, we can conclude that a wide range of queries, including shortest path queries can be expressed using the constructs that we defined. Further, the content based operators and the provenance structure graph based operators are combined to express several interesting queries that cannot be expressed using Datalog/SQL.

V. RELATED WORK

There has been lots of interest in provenance data management in the recent past. The initial works on provenance were tightly integrated with the application they considered [1, 5, 15]. The provenance community realized that it would be beneficial to have a uniform model for representing provenance and this led to the development of the Open Provenance Model (OPM) [20]. OPM describes how the provenance information can be represented in a general fashion; OPM also describes inference queries, such as the nodes that are ancestors or descendants of a node.

Most of the previous works on provenance also had some query mechanism, VisTrails [15] used the VisTrails query language called vtPQL; Kepler [1] provides a query language called QLP, which works on Kepler's proprietary provenance model; ZOOM [5] provides an interface for users to query provenance information similar to inference queries in OPM; Taverna [25] allows query specification using SPARQL query language; Karma [22] supports provenance queries using XPath and SQL. These works are tightly coupled with their underlying provenance representation mechanism, and hence is not general. Also, they are based on SQL or XPath kind of languages, and hence the expressiveness is limited by the expressiveness of these languages.

In [21], the authors propose an algebra for their provenance data model called provenir, which is defined using OWL-DL. The algebraic operators supported are provenance(), for obtaining the complete provenance (which can be expressed using ancestor/descendant operators); provenance_pathway(), which returns a subset of the information returned by provenance(), provenance_context(), which uses a user specified context (can be expressed using a combination of content-based operators for manipulating the context and ancestor/descendant operators). However, there are several other operators that we need for provenance metadata processing which are not supported. In [19], the authors consider scoping of provenance, because the entire

 ISBN: 978-1-61208-185-4

provenance might be overwhelming for the user. The authors consider the scenario where the user can explicitly exclude a part of the provenance. In our system, such scoping can be expressed either by developing a variation of the general selection operator that excludes part of the provenance, or by expressing it as first computing the complete provenance (using ancestor/descendant operators) and then excluding the portions that the user wants to exclude. In [26], the authors use the linking of data as supported by Linked Open Data (LOD) cloud; such context can be used for queries and for the interpretation of experimental data. In our work, such joins across multiple graphs can be expressed by unioning the graphs and then performing a join, or by selecting nodes/edges from one graph and using that as a context to select a subgraph of the second graph.

OPQL [16] introduces a query language that is directly defined over the Open Provenance Model (OPM). Here, the authors specify six types of graph patterns based on the patterns described in [11] and define three types of graph matching based on these graph patterns; the authors then define algebra operators for extracting sub-graphs that match a pattern, and for performing set operations such as union, intersection and difference. The implementation of OPQL uses SQL and hence their expressiveness is limited by what is expressible in SQL. Another work that considers querying data provenance is [13]. Here, the authors are concerned with data provenance; however, their query language is general and is applicable to workflow provenance as well. Their query language introduces XQuery style FOR-WHERE-RETURN expressions, with an additional INCLUDE PATH clause which specifies the paths that need to be included in the resulting graph. However, the WHERE clause is restricted so that shortest path kind of queries cannot be expressed, though it is possible to express ancestor/descendant queries. For manipulating graph data models, [11] specifies graph patterns and studies extracting sub-graphs as specified in the pattern.

The works described above [11, 13, 16] all manipulate graphs and return graphs as result. However, the expressiveness is limited to those expressible using Datalog/SQL; queries such as shortest paths are not considered. Also [11] is a query language for graphs in general, and not limited to provenance graph queries; some of the queries that may be interesting to general graphs, may not be interesting to provenance graphs.

Graph querying to a limited extent is supported by commercial database software such as PostgreSQL (http://pgfoundry.org/docman/view.php/1000262/505/README.txt) and MySQL (http://openquery.com/graph/doc). Here, graphs are represented as first class objects (not as tables), and an arbitrary set of graph operations are supported. For instance, MySQL supports shortest path, but does not support sub-graph isomorphism or path pattern matching. Also, they come up with SQL like syntax for all these operations, which we believe will be cumbersome if we want to perform a series of operations on a graph, as in a query plan. In short, we believe that support for graphs within databases is in its infancy, and most implementations are supporting only an arbitrary subset of graph operations. In our work, we provide a mechanism to query graphs that is general, where the users express their query as a workflow (thus are not limited by a text based language like SQL), and one query can express a series of operations to be performed on the graph.

In [2, 3], the authors consider querying of semantic associations from data represented using OWL. Here, the authors talk about given two nodes, how to determine the relationship between the nodes. One type of relationship referred to as ρ-pathAssociated (where there is a path between two nodes) can be expressed using our in-betweener operator, which is a type of the general selection operator. For other types of ρ-queries (relationships between nodes), we need to define other types of the general selection operator. However, note that in [2] the authors define a language exclusively for querying path information. However, for provenance data, we need more operators than just for querying the path information; we therefore use a workflow based approach, where the user specifies the query as a workflow consisting of any number of operators (as needed). One of the query constructs that we propose is a general selection operator for the structure graph of provenance; the only requirement is that this operator returns a sub-graph of the original graph; no restrictions are imposed on how the selection condition for nodes and edges are specified. Therefore, we are able to express a wide variety of queries including shortest path queries, sub-graph or path matching, ancestor, descendant, etc., while still maintaining a simple and elegant formalism of a selection operator.

VI. CONCLUSIONS AND FUTURE WORK

In this work, we studied algebraic constructs that can be used for provenance queries. Several interesting queries cannot be expressed using today's graph languages, or provenance query languages, as they rely on translating their queries into SQL/Datalog. We proposed two sets of algebraic constructs for querying provenance: content based operators manipulate the content (or annotation) of the nodes in a provenance graph; structure based operators manipulate the structure of a provenance graph. One of the powerful algebraic operators that we propose is a general selection operator that selects a sub-graph of the provenance structure graph, based on general restrictions on nodes and edges. An user expresses a query in our query model as a workflow by integrating these algebraic constructs. Our query model can express a wide range of interesting queries.

As part of future work, we are currently investigating optimization opportunities for the various operators and for provenance storage. Further, we are considering whether constructs that allow arbitrary addition of nodes and edges are useful for provenance queries, and how they can be supported.

ACKNOWLEDGMENT

We would like to acknowledge Yaobin Tang, who did initial work on provenance with us, and whose master's thesis led to many of the insights that we describe in this paper. We would also like to acknowledge the discussions

with the members of the University of Michigan, Flint, and Ann Arbor database groups, and discussions with Chunhyeok Lim and Shiyong Lu at Wayne State University, who are currently pursuing research on provenance. Last, but not the least, we would like to acknowledge NSF for partially supporting this research.

REFERENCES

[1] M. Ananad, S. Bowers, and B. Ludascher, Techniques for Efficiently Querying Scientific Workflow Provenance Graphs, In EDBT 2010, pp. 287 – 298.

[2] K. Anyanwu, A. Maduko, and A. P. Sheth, SPARQ2L: Towards Support for Subgraph Extraction Queries in RDF Databases, In WWW 2007, pp. 797 – 806.

[3] K. Anyanwu and A. P. Sheth, ρ -Queries: Enabling Querying for Semantic Associations on the Semantic Web, In WWW 2003, pp. 690 – 699.

[4] O. Benjelloun, A. D. Sarma, A. Y. Halevy, M. Theobald, and J. Widom, Databases with Uncertainty and Lineage, VLDB Journal, 17 (2), 2008, pp. 243 – 264.

[5] O. Biton, S. C. Boulakia, S. B. Davidson, and C. S. Hara, Querying and Managing Provenance through User Views in Scientific Workflows, In IEEE ICDE 2008, pp. 1072 – 1081.

[6] P. Buneman and W-C Tan, Provenance in Databases, In ACM SIGMOD 2007 (Tutorial), pp. 1171 – 1173.

[7] A. Chapman and H. V. Jagadish, Why Not? In ACM SIGMOD 2009, pp. 523 – 534.

[8] A. Chapman, H. V. Jagadish, and P. Ramanan, Efficient Provenance Storage, In ACM SIGMOD 2008, pp. 993 – 1006.

[9] Y. Cui and J. Widom, Lineage Tracing for General Data Warehouse Transformations, In VLDB Journal, 12 (1), 2003, pp. 41 – 58.

[10] T. G. Green, G. Karvounarakis, and V. Tannen, Provenance semirings, In ACM PODS 2007, pp. 31 – 40.

[11] H. He and A. K. Singh, Graphs-at-a-time: Query Language and Access Methods for Graph Databases, In ACM SIGMOD 2008, pp. 405 – 418.

[12] R. Ikeda and J. Widom, Panda: A System for Provenance and Data, In IEEE Data Engineering Bulletin, 33 (3), 2010, pp. 42 – 49.

[13] G. Karvounarakis, Z. G. Ives, and V. Tannen, Querying Data Provenance, In ACM SIGMOD 2010, pp. 951 – 962.

[14] D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. T. Silva, Bridging Workflow and Data Provenance Using Strong Links, In SSDBM 2010, pp. 397 – 415.

[15] D. Koop, C. E. Scheidegger, J. Freire, and C. T. Silva, The Provenance of Workflow Upgrades, In International Provenance and Annotation Workshop (IPAW), 2010, pp. 2 – 16.

[16] C. Lim, S. Lu, A. Chebotko, and F. Fatouhi, OPQL: A First OPM-Level Query Language for Scientific Workflow Provenance, In IEEE SCC 2011, pp. 136 – 143.

[17] C. Lim, S. Lu, A. Chebotko, and F. Fatouhi, Storing, Reasoning and Querying OPM-Compliant Scientific Workflow Provenance Using Relational Databases, In Future Generation Computer Systems, 27 (6), 2011, pp. 781 – 789.

[18] A. Meliou, W. Gatterbauer, J. Y. Halpern, C. Koch, K. F. Moore, and D. Suciu, Causality in Databases, In IEEE Data Engineering Bulletin, 33 (3), 2010, pp. 59 – 67.

[19] S. Miles, Electronically Querying for the Provenance of Entities, In IPAW 2006, pp. 184 – 192.

[20] The OPM Provenance Model (OPM), available at http://openprovenance.org/, retrieved: December, 2011.

[21] S. Sahoo, R. Barga, J. Goldstein, and A. Sheth, Provenance Algebra and Materialized View-based Provenance Management, Microsoft Research Technical Report, 2008, available at http://research.microsoft.com/apps/pubs/default.aspx?id=76523, retrieved: December, 2011.

[22] Y. Simmhan, B. Plale, and D. Gannon, Karma2: Provenance Management for Data Driven Workflows, International Journal of Web Services Research (IJWSR), 5 (2), 2008, pp. 1 – 22.

[23] Third Provenance Challenge, available at http://twiki.ipaw.info/bin/view/Challenge/ThirdProvenanceChallenge, retrieved: December, 2011.

[24] A. Woodruff and M. Stonebraker, Supporting Fine-Grained Data Lineage in a Data Visualization Environment, In IEEE ICDE, 1997, pp. 91 – 102.

[25] J. Zhao, C. Goble, R. Stevens, and D. Turi, Mining Taverna's Semantic Web of Provenance, In Concurrency and Computation: Practice and Experience, 20 (5), 2008, pp. 463 – 472.

[26] J. Zhao, S. S. Sahoo, P. Missier, A. P. Sheth, and C. A. Goble, Extending Semantic Provenance into the Web of Data, IEEE Internet Computing, 15 (1), 2011, pp. 40 – 48.

 ISBN: 978-1-61208-185-4