Logical Provenance in Data-Oriented Workflows* (Long Version)

Robert Ikeda
Stanford University
rmikeda@cs.stanford.edu

Akash Das Sarma
IIT Kanpur
akashds.iitk@gmail.com

Jennifer Widom
Stanford University
widom@cs.stanford.edu

Abstract

We consider the problem of defining, generating, and tracing provenance in data-oriented workflows, in which input data sets are processed by a graph of transformations to produce output results. We first give a new general definition of provenance for general transformations, introducing the notions of correctness, precision, and minimality. We then determine when properties such as correctness and minimality carry over from the individual transformations' provenance to the workflow provenance. We describe a simple logical-provenance specification language consisting of attribute mappings and filters. We provide algorithms for provenance tracing in workflows where logical provenance for each transformation is specified using our language. We consider logical provenance in the relational setting, showing that for a class of Select-Project-Join (SPJ) transformations, logical provenance specifications encode minimal provenance. We have built a prototype system supporting the features and algorithms presented in the paper, and we report a few preliminary experimental results.

1 Introduction

We address the problem of defining, generating, and tracing *provenance* in a setting of *data-oriented workflows*. In the workflow setting that we consider, *input data sets* are batch-processed through a graph of *transformations*, producing *intermediate data sets* and finally *output data sets*. In general, provenance retains information about how data propagates through the workflow.

Although existing systems such as [24, 27] are useful for building and executing workflows, capturing and exploiting provenance enables additional useful functionality in a workflow setting. For example, since provenance captures where data came from and how

*This work is supported by the National Science Foundation (IIS-0904497) and a KAUST research grant.

it is processed through the workflow, it can be very useful in supporting *debugging* and *drill-down*.

- **Debugging:** Workflows can have errors, either in the input data or the transformations. Finding such errors can be difficult and time-consuming—it may involve manually stepping through portions of the workflow on subsets of the data.
- **Drill-down:** Given a particular output element (or subset), a deeper understanding of how that element was generated may yield additional insights. Finding the relevant input data and processing steps for a specific output element is a feature not provided in most workflow systems.

There has been a great deal of past work on provenance in workflows. However, most approaches either track *coarse-grained* (schema and/or transformation level) provenance (e.g., [8, 20]), which is insufficient to support element-level debugging and drill-down, or they track the provenance of individual data elements with *physical provenance*. Physical provenance requires each output element to be annotated with some type of identifier for the contributing input elements (e.g., [10, 26]).

We support debugging and drill-down using *logical provenance*—provenance information stored at the transformation level. For many transformations, including a large subset of SQL, logical provenance can be derived automatically from the transformation's specification, and it captures exactly the same information as physical provenance but in a much more compact fashion (one specification per transformation rather than one per data element). Furthermore, for these transformations, logical provenance incurs no overhead at workflow-execution time, whereas physical provenance requires at a minimum the capture and storage of pointers. For some transformations, we must "augment" the output to support logical provenance. In the worst case, the augmenting process unavoidably degenerates to the equivalent of storing physical provenance.

The contributions of this paper are as follows:

- **Foundations of provenance** (Section 2). We give a new general definition of provenance, introducing the notions of *correctness*, *precision*, and *minimality*.
- **Theoretical properties of workflow provenance** (Section 3). Given a workflow with provenance defined for each of its transformations, we identify when provenance properties such as correctness and minimality carry over from individual transformations to the workflow as a whole.
- **Logical provenance specifications for transformations** (Section 4). We describe a simple logical-provenance specification language consisting of *attribute mappings* and *filters*.
- **Algorithms for provenance tracing** (Section 5). We provide an algorithm for provenance tracing in workflows where logical provenance for each transformation is specified using our language. In our algorithm we perform provenance tracing at the schema level to the extent possible, although eventually accessing the data obviously is required.

- **Logical provenance for relational transformations** (Section 6). We consider logical provenance in the relational setting, showing that for a class of *Select-Project-Join* (SPJ) *transformations*, logical provenance specifications encode minimal provenance.
- An **implementation** of the logical-provenance approach as part of the *Panda* (for *Provenance And Data*) system at Stanford (Section 7). Panda permits arbitrary data-oriented workflows with each transformation specified in either SQL or in Python [1]. We describe how Panda generates logical provenance specifications and executes our tracing algorithms. We also present some performance results (Section 8).

The remainder of this section presents a running example and surveys related work. In Section 9 we conclude and propose future directions.

1.1 Running Example

Our example is obviously unrealistic, crafted to demonstrate the major features of our approach in a compact fashion. We represent the data sets in our example as simple tables, although in general our approach only requires that each data set is a collection of *elements*, and there are some designated *attributes* that are present in each data element. (The remainder of each element can be structured arbitrarily.)

Consider "WebShop," an online reseller. WebShop periodically runs a workflow, shown in Figure 1, to calculate the total profits generated from sales for items that WebShop sells. The workflow's input data sets are:

- CustData(cust-id, country, activity_log), where activity_log is a free-text field containing customer activity such as clicks and purchases.
- ItemData(item-id, brand, type, price, supplier_info), where supplier_info is a free-text field containing information including cost from supplier.

The workflow involves the following transformations:

- Transformation **Extract** extracts from the activity_log attribute in CustData the items that each customer purchased, producing table CustSales(cust-id, country, item-id, quantity), abbreviated CS.
- Transformation **CalcProfit** calculates from the price and supplier_info attributes, for each item in ItemData, the profit made per item sold, producing table ItemProfit(item-id, brand, type, profit_per_item), abbreviated IP.
- Transformation **JoinAgg** joins tables CustSales and ItemProfit on attribute item-id, aggregating for each (item-id, country) pair the total profit from the item's sales in that country, producing table ItemCountryProfit(item-id, country, brand, type, profit), abbreviated IC.
- Transformation **Filter** filters ItemCountryProfit, selecting tuples with type equal to 'laptop', producing table LaptopProfit(item-id, country, brand, profit), abbreviated LP.[1]

[1]We assume intermediate data set ItemCountryProfit may be processed by other transformations as well; otherwise the workflow would of course filter for laptops much earlier.

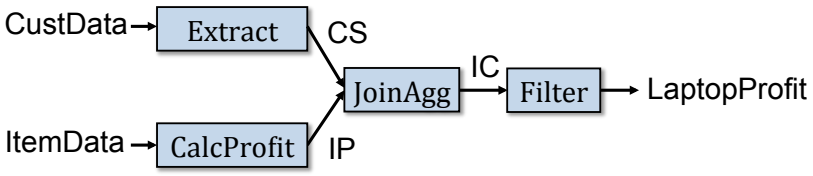

Figure 1: Profit calculation workflow.

CustData

cust-id	country	activity_log	
C1	France	<*free text*>	(1)
C2	Germany	<*free text*>	(2)
C3	France	<*free text*>	(3)

ItemData

item-id	brand	type	price	supplier_info	
I1	HP	laptop	700	<*free text*>	(1)
I2	Sony	tablet	550	<*free text*>	(2)
I3	Sony	laptop	800	<*free text*>	(3)

CustSales (CS)

cust-id	country	item-id	quantity	
C1	France	I1	5	(1)
C1	France	I3	7	(2)
C2	Germany	I1	6	(3)
C2	Germany	I2	4	(4)
C3	France	I3	8	(5)

ItemProfit (IP)

item-id	brand	type	profit_per_item	
I1	HP	laptop	120	(1)
I2	Sony	tablet	200	(2)
I3	Sony	laptop	10	(3)

ItemCountryProfit (IC)

item-id	country	brand	type	profit	
I1	France	HP	laptop	600	(1)
I1	Germany	HP	laptop	720	(2)
I2	Germany	Sony	tablet	800	(3)
I3	France	Sony	laptop	150	(4)

LaptopProfit (LP)

item-id	country	brand	profit	
I1	France	HP	600	(1)
I1	Germany	HP	720	(2)
I3	France	Sony	150	(3)

Figure 2: Profit calculation workflow, sample data.

Figure 2 shows sample input data sets along with all intermediate data, and finally output table LaptopProfit.

As a simple example of how provenance tracing can be useful for debugging, suppose we wanted to see why French profits on laptop sales with item-id I3 were so low. By tracing the provenance of LaptopProfit element (3) back through the workflow to ItemData, we discover that the profit per laptop (10, which may seem low) was extracted from the free-text field in ItemData element (3), suggesting that either the free-text field contains erroneous data or that the transformation **CalcProfit** has a bug.

To support provenance tracing, we propose a simple logical-provenance specification language consisting of *attribute mappings* and *filters*. As a straightforward example, consider transformation **Filter**. One attribute mapping that holds for this transformation is (IC.item-id, IC.country, IC.brand) $\leftrightarrow$ (LP.item-id, LP.country, LP.brand), which captures the fact that the output subset of LP with particular (item-id, country, brand) values corresponds to the input subset in IC with the same (item-id, country, brand) values. For this transformation, a filter `type='laptop'` also holds, indicating that output tuples in LP are never affected by elements in IC except those satisfying `type='laptop'`. Logical provenance specifications can often capture exactly the same element-level provenance information as physical provenance but in a much more compact fashion, and without the overhead of capturing and storing IDs or pointers. For transformations that are expressed as SQL queries, in most cases, logical provenance specifications can be generated through syntactic analysis of the SQL code; details are given in Section 7.

To provide the foundations for logical provenance, we give a new general definition of provenance in Section 2. As an example to illustrate the key components of our definition, consider CustSales element (2), which is in the output of transformation **Extract** applied to input set CustData. If we wanted a subset of CustData that explains why CustSales element (2) was output by transformation **Extract**, we could select elements (1) and (3) from CustData, since we know that the French records in the output correspond to French records in the input. However, even though this provenance is *correct*, it includes extra information; only CustData element (1) truly derives the output element of interest. Provenance containing only CustData element (1) is more *precise* than the previous provenance, and in fact, can be shown to be *minimal*; details are in Section 2.

In addition to reducing the overhead of provenance capture, logical provenance can often be combined across transformations, enabling more efficient tracing than the straightforward approach of tracing provenance through each step of intermediate data. As an example, we can automatically combine the logical provenance generated for **JoinAgg** and **Filter**, obtaining an attribute mapping between **JoinAgg**'s input data set CustSales (CS) and **Filter**'s output set LaptopProfit (LP): (CS.item-id, CS.country) $\leftrightarrow$ (LP.item-id, LP.country). (Although this example's attribute mappings only use basic equalities, we discuss more general attribute mappings in Section 4.) Using the composite logical provenance, we trace provenance directly from LaptopProfit to CustSales, bypassing intermediate data set ItemCountryProfit. An algorithm for combining logical provenance across transformations is given in Section 5.

1.2 Related Work

There has been a large body of work in lineage and provenance over the past two decades, surveyed in, e.g., [9, 13, 29]. As mentioned earlier, in contrast to our logical provenance, coarse-grained (schema and/or transformation level) provenance (e.g., [8, 20]) is usually insufficient to support element-level debugging and drill-down. Physical provenance can support debugging and drill-down, but requires each output element to be annotated with some type of identifier for the contributing input elements (e.g., [10, 26]).

Provenance models for relational and semistructured transformations are presented in, e.g., [6, 7, 11, 16, 17, 19, 25, 31]. Reference [11] contains a notion called *why-provenance*, which contains all *witnesses* (combinations of input elements) that produce a given output element. Our notion of minimal provenance is a generalization of why-provenance; for the special case of monotonic transformations, minimal provenance is equal to the union of all witnesses (Section 3.3). In contrast to [16], our work defines notions of correctness and minimality for provenance that apply to general transformations, not just relational transformations. Reference [25] defines a notion related to provenance called *causality*, which captures not only the input elements that contribute to an output element, but also a measure of how responsible each input element is for producing the output element; the major results of [25] are restricted to conjunctive queries. Reference [7] considers the provenance of individual attribute values (*where-provenance*), while our work focuses on the provenance of data elements. Reference [17] captures provenance by augmenting relational transformations via query rewrite, in effect annotating output elements with provenance information. Our approach in some cases must resort to augmentation, but we attempt as much as possible to capture provenance information at the transformation level.

Provenance in the context of schema mappings is studied in, e.g., [14, 18, 31]. References [14] and [18] both consider schema mapping languages that cannot express general transformations (e.g., aggregation). Reference [31] considers a different type of provenance that consists of schema elements and transformations as opposed to input data elements.

Provenance in the data-oriented workflow setting is considered by [3, 4, 5, 12, 15, 21, 22, 23, 27, 32], among others. Reference [21] describes a demonstration scenario for the system presented in this paper, but does not provide any foundations, algorithms, or technical results. Reference [4] uses Pig Latin [28] to express the functionality of workflow modules, from which provenance information is generated, but unlike our general approach, provenance definitions are tightly coupled with the specific Pig Latin operations. Reference [12] presents techniques for reducing the space overhead of provenance storage, while our logical provenance uses transformation properties to avoid storing physical provenance altogether. Reference [32] requires the workflow creator to specify transformation *inverses* for each transformation from which provenance can be computed; it has no support for automatically computing logical provenance for well-understood transformations, such as relational ones.

Reference [33] describes tracking provenance for arbitrary functions by instrumenting the program binary that implements the function. Since the approach in [33] involves capturing provenance based on a particular execution path, its definition of correct provenance may be missing relevant input elements. Also, by analyzing the properties of workflow provenance, we will see that an execution-based definition of provenance may include input elements that actually have no impact on the final output.

Reference [15] considers general transformations, providing a hierarchy of transformation types relevant to provenance; each transformation is placed in the hierarchy by its creator to make provenance tracing efficient. Provenance in [15] is defined separately for each transformation type, while the definitions for provenance given in this paper are uni-

fied across all transformations. Also, while [15] allows acyclic graphs of transformations, it does not investigate when properties such as correctness and minimality carry over from individual transformations to a composite workflow.

2 Foundations of Provenance

Let a *data set* be any set of data *elements*. We assume each data set has some predefined *attributes* that are present in each element. Elements may contain arbitrary additional data, which we do not use for provenance. Attributes will not be used until Section 4, when we present our language for specifying logical provenance. Our formal definitions for provenance in this section and Section 3 are independent of element structure.

A *transformation* is any procedure that takes one or more data sets as input and produces one or more data sets as output. Recall from the running example that transformations are not always relational queries. A *workflow* is a directed acyclic graph, where nodes denote transformations, and edges denote the flow of the data sets that are input to and output from the transformations.

For now, we consider a single transformation T with a single input I and multiple outputs $O_1, \ldots, O_r$. Let $O = O_1 \cup \ldots \cup O_r$. Let $O = T(I)$ be a transformation instance, denoting the application of transformation T to input set I, obtaining output set O. (We will generalize to multiple inputs in Section 2.4, and we will assemble transformations into workflows in Section 3.) We assume $T(\emptyset) = \emptyset$. Our goal is to specify data-level provenance relationships between output elements in O and input elements in I. Specifically, given an output element $o \in O$, we would like to know the input subset $I^* \subseteq I$ that "produced" o. Defining this notion of provenance formally for general transformations can be challenging, as evidenced by a variety of definitions in the literature [13, 15, 16].

Here, we give a new general definition of provenance for general transformations, introducing the notions of *correctness*, *precision*, and *minimality*. Intuitively, correct provenance $I^* \subseteq I$ must contain the "essence" of what derives o. Correct provenance typically is not unique, so we say correct provenance I^* is more precise than I^{**} if $I^* \subset I^{**}$. As we will see, it turns out that there always exists a most precise provenance, which we refer to as the minimal provenance.

2.1 Correctness

Recall that correct provenance for an output element $o \in O$ should contain the "essence" of what derives o. Here is our formal definition:

Definition 2.1 (Correctness) Let $O = T(I)$ be a transformation instance. Consider an output element $o \in O$. Provenance $I^* \subseteq I$ is *correct* for o with respect to T if:

- For any $I' \subseteq I$, $o \in T(I')$ if and only if $o \in T(I' \cap I^*)$. □

Intuitively, this definition says that I^* is correct provenance if given any input subset I', we only need to consider $T(I' \cap I^*)$ to determine whether or not $o \in T(I')$. In other words,

the only input elements in I' that could contribute to o are those that are also in I^*. Note that setting $I' = I$ shows that if I^* is correct for o, then $o \in T(I^*)$.

Example 2.1 Let $o = \mathsf{CustSales}(1)$, i.e., the first element from data set CustSales in Figure 2. Let $I^* = \{\mathsf{CustData}(1), \mathsf{CustData}(3)\}$. I^* is correct provenance although we will soon see that it is not minimal. Intuitively, we would expect I^* to be correct for o, since I^* contains the one element $\mathsf{CustData}(1)$ that derives o. To check formally that I^* is correct for o with respect to transformation **Extract**, we can simply verify that for each of the eight subsets $I' \subseteq \mathsf{CustData}$, the condition in Definition 2.1 is satisfied. □

2.2 Precision

Correctness alone does not capture the intuitive notion of provenance. For example, setting I^* equal to the entire input set I always yields correct provenance, but obviously this choice of I^* rarely gives useful information. Given two different subsets of I that are both correct provenance for an output element o, we prefer the smaller subset, since the smaller subset tells us more about what actually contributed to o.

Definition 2.2 (Precision) Let $O = T(I)$ and let $o \in O$. Suppose subsets $I^* \subseteq I$ and $I^{**} \subseteq I$ are both correct provenance for o with respect to T. I^* is *at least as precise* as I^{**} if $I^* \subseteq I^{**}$. I^* is *more precise* than I^{**} if $I^* \subset I^{**}$. □

Example 2.2 Consider again $o = \mathsf{CustSales}(1)$. Let $I^* = \{\mathsf{CustData}(1)\}$. It is easy to verify that I^* is correct provenance for o with respect to transformation **Extract**. Furthermore, since $I^* \subset \{\mathsf{CustData}(1), \mathsf{CustData}(3)\}$, I^* is more precise than the provenance from Example 2.1. □

2.3 Minimality

It turns out that there always exists a single most precise provenance, which we refer to as the minimal provenance. Intuitively, the minimal provenance captures "the essence" of what derives o.

Definition 2.3 (Minimality) Let $O = T(I)$ and let $o \in O$. Suppose subset $I^* \subseteq I$ is correct provenance for o with respect to T. I^* is *minimal* for o with respect to T if there does not exist provenance I^{**} that is correct and more precise than I^*. □

In Section 2.4, we will give an example of minimal provenance after we have defined minimality for multi-input transformations. We now show that there always exists a unique minimal provenance.

Theorem 2.1 (Unique Minimal Provenance) Let $O = T(I)$ and let $o \in O$. Let $I_1^*, \ldots, I_n^*$ be all of the correct provenances for o with respect to T. Let $I^M = I_1^* \cap \ldots \cap I_n^*$.

I^M is correct and minimal, and there is no other correct and minimal provenance for o with respect to T.

Proof. Since I is always correct provenance for o, there always exists at least some subset of I that is correct provenance for o. And since I is finite, we can take the intersection of all subsets of I that are correct provenance for o. Thus I^M is well-defined.

We show that I^M is correct. To do so, we will show that the intersection of any two correct provenances is correct provenance, from which the correctness of I^M follows. Let $I^* \subseteq I$ and $I^{**} \subseteq I$ both be correct provenance for o. We show that $I^* \cap I^{**}$ is correct provenance for o. We need to show that for any $I' \subseteq I$, $o \in T(I')$ if and only if $o \in T(I' \cap (I^* \cap I^{**}))$. Since I^* is correct, we know that $o \in T(I')$ if and only if $o \in T(I' \cap I^*)$. Since I^{**} is correct, we know that $o \in T(I' \cap I^*)$ if and only if $o \in T((I' \cap I^*) \cap I^{**})) = T(I' \cap (I^* \cap I^{**}))$. Thus, $I^* \cap I^{**}$ is correct provenance for o.

We now show that I^M is minimal by contradiction. Suppose there existed correct provenance I^{**} for o such that $I^{**} \subset I^M$. By the definition of I^M, since I^{**} is correct provenance, we know that $I^{**} = I_j^*$ for some j and thus $I^M = I_1^* \cap \ldots \cap I_n^* \subseteq I_j^* = I^{**}$, a contradiction.

We now show that I^M is unique. Let I^{**} be minimal provenance. Since I^M is minimal, we know $I^M \subseteq I^{**}$. Since I^{**} is minimal, we know $I^{**} \subseteq I^M$. Thus, $I^{**} = I^M$. □

2.4 Multiple Inputs

The definitions for correctness, precision, and minimality generalize naturally to multiple input sets.

Definition 2.4 (Correctness Definition) Let $O = T(I_1, \ldots, I_m)$ and let $o \in O$. Provenance $\langle I_1^*, \ldots, I_m^* \rangle$, $I_1^* \subseteq I_1, \ldots, I_m^* \subseteq I_m$ is *correct* for o with respect to T if:

- For any $\langle I_1', \ldots, I_m' \rangle$ such that $I_1' \subseteq I_1, \ldots, I_m' \subseteq I_m$, $o \in T(I_1', \ldots, I_m')$ if and only if $o \in T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*)$. □

Definition 2.5 (Precision Definition) Let $O = T(I_1, \ldots, I_m)$ and let $o \in O$. Suppose $\langle I_1^*, \ldots, I_m^* \rangle$ and $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ are both correct provenance for o with respect to T. $\langle I_1^*, \ldots, I_m^* \rangle$ is *at least as precise* as $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ if $I_1^* \subseteq I_1^{**}, \ldots, I_m^* \subseteq I_m^{**}$. $\langle I_1^*, \ldots, I_m^* \rangle$ is *more precise* than $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ if $I_1^* \subseteq I_1^{**}, \ldots, I_m^* \subseteq I_m^{**}$ and there exists some i for which $I_i^* \subset I_i^{**}$. □

Definition 2.6 (Minimality Definition) Let $O = T(I_1, \ldots, I_m)$ and let $o \in O$. Suppose $\langle I_1^*, \ldots, I_m^* \rangle$ is correct provenance for o with respect to T. $\langle I_1^*, \ldots, I_m^* \rangle$ is *minimal* for o with respect to T if there does not exist provenance $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ that is correct and more precise than $\langle I_1^*, \ldots, I_m^* \rangle$. □

Theorem 2.2 (Unique Minimal Provenance) Let $O = T(I_1, \ldots, I_m)$ and let $o \in O$. Let $\langle I_1^{1*}, \ldots, I_m^{1*} \rangle, \ldots, \langle I_1^{n*}, \ldots, I_m^{n*} \rangle$ be all of the correct provenances for o with respect to

T. Let $I^M = \langle (I_1^{1*} \cap \ldots \cap I_1^{n*}), \ldots, (I_m^{1*} \cap \ldots \cap I_m^{n*}) \rangle$. I^M is correct and minimal, and there is no other correct and minimal provenance for o with respect to T.

Proof. Since $\langle I_1, \ldots, I_m \rangle$ is always correct provenance for o, there always exists at least some correct provenance for o. And since $\langle I_1, \ldots, I_m \rangle$ is finite, we can take the intersection of all subsets of $\langle I_1, \ldots, I_m \rangle$ that are correct provenance for o. Thus I^M is well-defined.

We show that I^M is correct. To do so, we will show that the intersection of any two correct provenances is correct provenance, from which the correctness of I^M follows. Let $\langle I_1^*, \ldots, I_m^* \rangle$ and $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ both be correct provenance for o. We show that $\langle (I_1^* \cap I_1^{**}), \ldots, (I_m^* \cap I_m^{**}) \rangle$ is correct provenance for o. We need to show that for any $\langle I_1', \ldots, I_m' \rangle$ such that $I_1' \subseteq I_1, \ldots, I_m' \subseteq I_m$, $o \in T(I')$ if and only if $o \in T((I_1' \cap (I_1^* \cap I_1^{**})), \ldots, (I' \cap (I_m^* \cap I_m^{**})))$. Since $\langle I_1^*, \ldots, I_m^* \rangle$ is correct, we know that $o \in T(I_1', \ldots, I_m')$ if and only if $o \in T((I_1' \cap I_1^*), \ldots, (I_m' \cap I_m^*))$. Since $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ is correct, we know that $o \in T((I_1' \cap I_1^*), \ldots, (I_m' \cap I_m^*))$ if and only if $o \in T(((I_1' \cap I_1^*) \cap I_1^{**}), \ldots, ((I_m' \cap I_m^*) \cap I_m^{**})) = T((I_1' \cap (I_1^* \cap I_1^{**})), \ldots, (I_m' \cap (I_m^* \cap I_m^{**})))$. Thus, $\langle (I_1^* \cap I_1^{**}), \ldots, (I_m^* \cap I_m^{**}) \rangle$ is correct provenance for o.

We now show that $I^M = \langle I_1^M, \ldots, I_m^M \rangle$ is minimal by contradiction. Suppose there existed correct provenance $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ for o such that $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ is more precise than I^M. By the definition of I^M, since $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ is correct provenance and $\langle I_1^M, \ldots, I_m^M \rangle = \langle (I_1^{1*} \cap \ldots \cap I_1^{n*}), \ldots, (I_m^{1*} \cap \ldots \cap I_m^{n*}) \rangle$, we know that $I_1^M \subseteq I_1^{**}, \ldots, I_m^M \subseteq I_m^{**}$, contradicting the assumption that $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ is more precise than $\langle I_1^M, \ldots, I_m^M \rangle$.

We now show that $I^M = \langle I_1^M, \ldots, I_m^M \rangle$ is unique. Let $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ be minimal provenance. Since $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ is correct provenance, by definition of I^M we know that $I_1^M \subseteq I_1^{**}, \ldots, I_m^M \subseteq I_m^{**}$. Since $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ is minimal, we know that $\langle I_1^M, \ldots, I_m^M \rangle$ cannot be more precise than $\langle I_1^{**}, \ldots, I_m^{**} \rangle$, implying that $I_1^M \not\subset I_1^{**}, \ldots, I_m^M \not\subset I_m^{**}$. Thus, $\langle I_1^{**}, \ldots, I_m^{**} \rangle = \langle I_1^M, \ldots, I_m^M \rangle$. □

Example 2.3 Consider $o = \mathsf{ItemCountryProfit}(4)$. Minimal provenance for o with respect to transformation **JoinAgg** is $\langle \mathsf{CS}^*, \mathsf{IP}^* \rangle$, where $\mathsf{CS}^* = \{\mathsf{CustSales}(2), \mathsf{CustSales}(5)\}$ and $\mathsf{IP}^* = \{\mathsf{ItemProfit}(3)\}$. □

3 Workflow Provenance

We discuss the theoretical properties of workflow provenance, identifying when provenance properties such as correctness and minimality carry over from individual transformations to the workflow as a whole.

Consider any transformations T_1 and T_2. The *composition* $T_1 \circ T_2$ of the two transformations is a transformation that first applies T_1 to an input data set I_1 to obtain *intermediate* data set I_2. It then applies T_2 to I_2 to obtain output data set O.

Composition is associative, so we denote the linear composition of n transformations as $T_1 \circ T_2 \circ \cdots \circ T_n$. We refer to such a composition as a *data-oriented workflow*, or *workflow* for short. In Section 3.5, we extend our formalism to cover workflows where transformations

may have multiple input and output data sets, allowing workflows to be arbitrary DAGs. (We do not consider cycles in this paper.)

Consider a workflow $T_1 \circ T_2 \circ \ldots \circ T_n$. A *workflow instance* is the application of the workflow to an input I_1. Let $I_{i+1} = T_i(I_i)$ for $i = 1..n$. The final output data set is I_{n+1}. We denote this workflow instance as $(T_1 \circ T_2 \circ \ldots \circ T_n)(I_1) = I_{n+1}$.

Suppose each of the transformations T_i includes a provenance specification. For each element $o \in I_{i+1}$, let $P_{T_i}(o) \subseteq I_i$ denote the provenance of o with respect to T_i. Then we can define workflow provenance in the intuitive recursive way as follows.

Definition 3.1 (Workflow Provenance) Let $W = T_1 \circ T_2 \circ \ldots \circ T_n$. Consider a workflow instance $(T_1 \circ T_2 \circ \ldots \circ T_n)(I_1) = I_{n+1}$, with initial input I_1. Let e be any data element involved in the workflow—input, intermediate, or output. The *workflow provenance* of e in W, denoted $P_W(e)$, is an input subset $I_1^* \subseteq I_1$. If e is an initial input element, i.e., $e \in I_1$, then $P_W(e) = \{e\}$. Otherwise, let T be the transformation that output e. Let $P_T(e)$ be the one-level provenance of e with respect to T. Then $P_W(e) = \bigcup_{e' \in P_T(e)} P_W(e')$. □

Having defined workflow provenance in the intuitive way, we are interested in determining when properties such as correctness and minimality carry over from the individual transformations' provenance to the workflow provenance. We first define some transformation properties.

Definition 3.2 (Monotonicity) A transformation T is *monotonic* if for any input sets I and I', if $I \subseteq I'$, then $T(I) \subseteq T(I')$. □

In our running example of Section 1.1, all of the transformations except **JoinAgg** (technically requiring the extended formalism in Section 3.5) are monotonic.

Definition 3.3 (1-m Transformations) A transformation T is *1-m* if for any input set I, each output element $o \in T(I)$ has exactly one input element in its minimal provenance. □

Definition 3.4 (m-1 Transformations) A transformation T is *m-1* if for any input set I, each input element $e \in I$ is in at most one output element's minimal provenance. □

Let us now introduce a provenance property weaker than correctness, which we call *weak correctness*. We will see cases where in a workflow we can guarantee weak correctness but not correctness.

Definition 3.5 (Weak Correctness Definition) Let $O = T(I)$ be a transformation instance and let $o \in O$. We say that provenance $I^* \subseteq I$ is *weakly correct* for o with respect to T if:

- For any $I' \subseteq I$, if $I^* \subseteq I' \subseteq I$, then $o \in T(I')$. □

This definition states that I^* is weakly correct provenance for o with respect to T if T produces o when given any superset of I^* as input. Note that for any monotonic transformation T, if $o \in T(I^*)$, then I^* is weakly correct for o with respect to T.

As a simple example of how weak correctness and correctness differ, consider an instance of a deduplicating transformation. Given an output element, any of the corresponding duplicates from the input set would alone constitute weakly correct provenance. However, correct provenance must contain all of the duplicates.

3.1 Remainder of Section

We now outline the major results of this section. Let $W = T_1 \circ T_2 \circ \ldots \circ T_n$. Consider a workflow instance $(T_1 \circ T_2 \circ \ldots \circ T_n)(I_1) = I_{n+1}$ in which each of the transformations T_i includes a correct provenance specification.

- **Preservation of correctness** (Section 3.2). If all transformations T_i are monotonic, then W's workflow provenance is correct (Theorem 3.1). However, if there exists any nonmonotonic transformation in W, then W's workflow provenance is not necessarily correct (Example 3.1).
- **Preservation of minimality** (Section 3.3). If all transformations T_i are monotonic, we have minimal provenance at each transformation, $T_1, \ldots, T_j$ are m-1, and $T_{j+1}, \ldots, T_n$ are 1-m, then W's workflow provenance is minimal (Theorem 3.2). However, if all transformations T_i are monotonic and we have minimal provenance at each transformation (but we don't have the m-1 and 1-m properties of the previous statement), then W's provenance is not necessarily minimal (Example 3.2).
- **Preservation of weak correctness** (Section 3.4). If there exists at most one nonmonotonic transformation in W, then W's workflow provenance is weakly correct (Theorem 3.3). However, if there is more than one nonmonotonic transformation in W, then W's workflow provenance is not necessarily weakly correct (Example 3.3).

3.2 Preservation of Correctness

If we have correct provenance for each transformation in the workflow, and each transformation is monotonic, then the workflow provenance (as defined in Definition 3.1) is also correct.

Theorem 3.1 Let $W = T_1 \circ T_2 \circ \ldots \circ T_n$. Consider a workflow instance $(T_1 \circ T_2 \circ \ldots \circ T_n)(I_1) = I_{n+1}$ in which all of the transformations are monotonic. For each T_i and each element $e \in I_{i+1}$, let $P_{T_i}(e) \subseteq I_i$ be correct provenance for e with respect to T_i. Consider any output element $o \in I_{n+1}$. Workflow provenance $P_W(o)$ is correct for o with respect to W.

Proof. We prove by induction on n. The base case of $n = 1$ follows from the assumption. Now suppose the theorem holds for $n = k$ and consider the theorem for $n = k+1$. Consider $o \in I_{k+2}$. $P_W(o) = \bigcup_{e \in P_{T_{k+1}}(o)} P_W(e)$.

Figure 3: Counterexample for correct workflow provenance.

We prove by contradiction. Suppose $I_1^* = P_W(o)$ is not correct for o with respect to $W_{k+1} = T_1 \circ T_2 \circ \ldots \circ T_{k+1}$. Then there exists some $I_1' \subseteq I_1$ such that $W_{k+1}(I_1') \cap \{o\} \neq W_{k+1}(I_1' \cap I_1^*) \cap \{o\}$. Since each transformation in W_{k+1} is monotonic, W_{k+1} is monotonic. It then follows from W_{k+1}'s monotonicity and the above inequality that $o \in W_{k+1}(I_1')$ and $o \notin W_{k+1}(I_1' \cap I_1^*)$.

For any $e \in I_{k+1}^* = P_{T_{k+1}}(o)$, by the inductive hypothesis we know that $P_W(e)$ is correct for e with respect to $W_k = T_1 \circ T_2 \circ \ldots \circ T_k$. Since $P_W(e) \subseteq I_1^*$ for any $e \in I_{k+1}^*$, we know that $e \in W_k(I_1')$ if and only if $e \in W_k(I_1' \cap I_1^*)$, from which it follows that $W_k(I_1') \cap I_{k+1}^* = W_k(I_1' \cap I_1^*) \cap I_{k+1}^*$.

Recall that $o \in W_{k+1}(I_1') = T_{k+1}(W_k(I_1'))$. Since I_{k+1}^* is correct for o with respect to T_{k+1}, we then know that $o \in T_{k+1}(W_k(I_1') \cap I_{k+1}^*) = T_{k+1}(W_k(I_1' \cap I_1^*) \cap I_{k+1}^*)$. Since T_{k+1} is monotonic, it follows that $o \in T_{k+1}(W_k(I_1' \cap I_1^*)) = W_{k+1}(I_1' \cap I_1^*)$, a contradiction. Thus, I_1^* is correct. □

If we drop the assumption that all transformations are monotonic, we find an example demonstrating that workflow provenance is not necessarily correct.

Example 3.1 Consider workflow $W = \textbf{SingleStore} \circ \textbf{Projection}$ shown in Figure 3. The initial input data set SalesInfo contains country-sales pairs for each of a corporation's worldwide stores: $\mathsf{SalesInfo}(\mathsf{country}, \mathsf{sales}) = \{(France, 10), (Germany, 20), (Germany, 10)\}$. Transformation **SingleStore** retains sales information for stores in countries with only one store, producing intermediate data set SingleStoreCountries (abbreviated SSC): $\mathsf{SSC} = \textbf{SingleStore}(\mathsf{SalesInfo}) = \{(France, 10)\}$. The following provenance is correct: $P_{\textbf{SingleStore}}((France, 10)) = \{(France, 10)\} \subseteq \mathsf{SalesInfo}$. Transformation **Projection** projects away the country name, leaving only total sales for each of the stores in SSC: $\textbf{Projection}(\mathsf{SSC}) = \{10\}$. Note that **SingleStore** is not monotonic, but **Projection** is. The following provenance is correct: $P_{\textbf{Projection}}(10) = \{(France, 10)\}$. Let $o = 10 \in \mathsf{SalesNum}$. The workflow provenance $P_W(o)$ of o following Definition 3.1 is $\{(France, 10)\} \subseteq \mathsf{SalesInfo}$. However, the only correct provenance for o with respect to W is actually all of SalesInfo: $\{(France, 10), (Germany, 20), (Germany, 10)\}$. To see, for example, that $(Germany, 10)$ needs to be in correct provenance I^*, note that by setting $\mathsf{SalesInfo}' = \{(Germany, 10)\}$, we get $o \in W(\mathsf{SalesInfo}')$. However, if $(Germany, 10) \notin I^*$, then $o \notin W(\mathsf{SalesInfo}' \cap I^*)$. Thus, workflow provenance $P_W(o)$ is not correct. □

3.3 Preservation of Minimality

Although correctness carries over from the provenance of individual transformations to workflow provenance when all transformations are monotonic, we now show an example demonstrating that there does not exist such a guarantee for minimality.

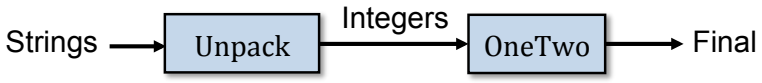

Figure 4: Counterexample for minimal workflow provenance.

Example 3.2 Consider workflow $W = \textbf{Unpack} \circ \textbf{OneTwo}$ shown in Figure 4. The input data set Strings contains two strings: {"1", "1, 2"}. Transformation **Unpack** finds all integers contained in Strings and removes duplicates, producing intermediate data set Integers $= \{1, 2\}$. We have minimal provenance for **Unpack**: $P_{\textbf{Unpack}}(1) = \{$"1", "1, 2"$\}$ and $P_{\textbf{Unpack}}(2) = \{$"1, 2"$\}$. Transformation **OneTwo** processes intermediate data set Integers as follows:

- If $1 \in$ Integers and $2 \in$ Integers, then **OneTwo**(Integers) $= \{$"success"$\}$
- Else, **OneTwo**(Integers) $= \emptyset$

In this workflow instance, Final = **OneTwo**(Integers) $= \{$"success"$\}$. The minimal provenance of "success" with respect to **OneTwo** is $P_{\textbf{OneTwo}}($"success"$) = \{1, 2\}$.

Note that **Unpack** and **OneTwo** are both monotonic. Let $o =$ "success" $\in$ Final. The workflow provenance $P_W(o)$ of o according to Definition 3.1 is {"1", "1, 2"}. However, {"1, 2"} is also correct provenance for o with respect to W, and it is minimal. Thus, workflow provenance in this example is not minimal. □

Intuitively, in the above example, workflow provenance is not minimal because it contains an input element e ("1" in our example) that can never actually influence whether the output element is produced. In other words, the uninfluential input element e's contribution is never sufficient to change the ultimate outcome, yet e is included in the provenance.

Let us identify a class of workflows for which the above situation does not occur. If a monotonic workflow consists of a set of m-1 transformations followed by a set of 1-m transformations, and we have minimal provenance for each transformation in the workflow, then workflow provenance is minimal.

Theorem 3.2 Let $W = T_1 \circ T_2 \circ \ldots \circ T_n$. Consider a workflow instance $(T_1 \circ T_2 \circ \ldots \circ T_n)(I_1) = I_{n+1}$ in which all of the transformations are monotonic, $T_1, \ldots, T_j$ are m-1 transformations, and $T_{j+1}, \ldots, T_n$ are 1-m transformations. For each T_i and each element $e \in I_{i+1}$, let $P_{T_i}(e) \subseteq I_i$ be minimal provenance for e with respect to T_i. Consider any output element $o \in I_{n+1}$. Workflow provenance $P_W(o)$ is minimal for o with respect to W. □

The proof of this theorem involves some lemmas. Before the lemmas, we define *minimal witnesses* [11].

Definition 3.6 (Minimal Witness) Let $O = T(I)$ be a transformation instance and let $o \in O$. Subset $I^w \subseteq I$ is a *minimal witness* of o with respect to T if: (1) $o \in T(I^w)$; and (2) for any proper subset $I' \subset I^w$, $o \notin T(I')$. □

As an example to illustrate the difference between minimal witnesses and minimal provenance, consider a transformation that eliminates duplicates. For any output element o,

each of the corresponding duplicates o in the input data set is a separate minimal witness, while the minimal provenance is the entire set of o's in the input. The following lemma generalizes this example, showing that for any output element o from a monotonic transformation, the union of all minimal witnesses of o is equal to o's minimal provenance.

Lemma 3.1 Let $O = T(I)$ be an instance of a monotonic transformation, and let $o \in O$. Let $I_1^w, \ldots, I_m^w$ be all of the minimal witnesses of o with respect to T, and let I^* be the minimal provenance of o with respect to T. Then $I^* = I_1^w \cup \ldots \cup I_m^w$.

Proof of Lemma 3.1 We first show that $I^* \supseteq I_1^w \cup \ldots \cup I_m^w$. Given any element $e \in I_1^w \cup \ldots \cup I_m^w$, we show $e \in I^*$ by contradiction. Suppose $e \notin I^*$. Since $e \in I_1^w \cup \ldots \cup I_m^w$, e is in some I_j^w. Since I_j^w is a minimal witness of o, I^* is correct for o, and T is monotonic, we know that $\{o\} = (T(I_j^w) \cap \{o\}) = (T(I_j^w \cap I^*) \cap \{o\}) \subseteq (T(I_j^w - \{e\}) \cap \{o\})$. Since I_j^w is a minimal witness, $o \notin T(I_j^w - \{e\})$, thus $(T(I_j^w - \{e\}) \cap \{o\}) = \emptyset$, a contradiction. Thus, $e \in I^*$, implying that $I^* \supseteq I_1^w \cup \ldots \cup I_m^w$.

Now we show that $I^* \subseteq I_1^w \cup \ldots \cup I_m^w$. Given any element $e \in I^*$, we show $e \in I_1^w \cup \ldots \cup I_m^w$. Since e is in the minimal provenance I^* of o, we know that $I^* - \{e\}$ is not correct provenance, implying that there exists some subset $I' \subseteq I$ such that $o \in T(I')$ but $o \notin T(I' \cap (I^* - \{e\})) = T(I' \cap I^* - \{e\})$. Since I^* is correct, we know that $o \in T(I' \cap I^*)$. Consider all subsets of $I' \cap I^*$. We know at least one subset, namely the entire $I' \cap I^*$, produces o. Take the smallest subset I'' (pick arbitrarily if there is a tie) that produces o. It must contain e because T is monotonic and $o \notin T(I' \cap I^* - \{e\})$. Also, no subset of I'' can produce o, since we picked the minimal subset. Thus, I'' is equal to some minimal witness I_j^w, from which it follows that $e \in I_1^w \cup \ldots \cup I_m^w$. We have shown that $I^* \subseteq I_1^w \cup \ldots \cup I_m^w$. Together with $I^* \supseteq I_1^w \cup \ldots \cup I_m^w$, it follows that $I^* = I_1^w \cup \ldots \cup I_m^w$. □

Lemma 3.2 Let $W = T_1 \circ \ldots \circ T_n$ where each T_i is a monotonic m-1 transformation. Then W is also an m-1 transformation.

Proof of Lemma 3.2 We prove by induction on n. For $n = 1$, the lemma follows from the assumption. Now suppose the lemma holds for $n = k$ and consider the lemma for $n = k + 1$. Since the lemma holds for $n = k$, we know $W_k = T_1 \circ \ldots \circ T_k$ is m-1.

For any element $o \in I_{k+2}$, let $P_{T_{k+1}}(o) \subseteq I_{k+1}$ be the minimal provenance of o with respect to T_{k+1}. For any element $e_{k+1} \in I_{k+1}$, let $P_{W_k}(e_{k+1}) \subseteq I_1$ be the minimal provenance of e_{k+1} with respect to W_k. Let $P_{W_{k+1}}(o) = \cup_{e \in P_{T_{k+1}}(o)} P_{W_k}(e)$ be the workflow provenance of o in $W_{k+1} = W_k \circ T_{k+1}$. Then by Theorem 3.1, since we have minimal (and thus correct) provenance for W_k and T_{k+1}, $P_{W_{k+1}}(o)$ is correct for o with respect to W_{k+1}.

Given any $e_1 \in I_1$, since e_1 is in the minimal provenance of at most one element $e_{k+1} \in I_{k+1}$, and e_{k+1} is in the minimal provenance of at most one element $o \in I_{k+2}$, we know that e_1 is in $P_{W_{k+1}}(o)$ for at most one element $o \in I_{k+2}$. Let $P(o) \subseteq I_1$ be the minimal provenance of o with respect to W_{k+1}. Since $P_{W_{k+1}}(o)$ is correct, $P_{W_{k+1}}(o) \supseteq P(o)$ for all $o \in I_{k+2}$. Thus, e_1 is in the minimal provenance $P(o)$ for at most one element $o \in I_{k+2}$, i.e., W_{k+1} is m-1. □

Lemma 3.3 Let $W = T_1 \circ \ldots \circ T_n$. Consider a workflow instance $(T_1 \circ T_2 \circ \ldots \circ T_n)(I_1) = I_{n+1}$ where each T_i is a monotonic m-1 transformation. Suppose we have minimal provenance at each transformation. Consider any output element $o \in I_{n+1}$. Workflow provenance $P_W(o)$ is minimal for o with respect to W.

Proof of Lemma 3.3 We prove by induction on n. For $n = 1$ the lemma follows from the assumption. Now suppose the lemma holds for $n = k$ and consider the lemma for $n = k+1$. Since the lemma holds for $n = k$, for all elements $e \in I_{k+1}, P_W(e)$ is minimal for e with respect to $W_k = T_1 \circ \ldots \circ T_k$.

Consider $o \in I_{k+2}$. $P_W(o) = \bigcup_{e \in P_{T_{k+1}}(o)} P_W(e)$. To show workflow provenance $I_1^* = P_W(o)$ is minimal, we need to show that for all elements $e_1 \in I_1^*$, $I_1^* - \{e_1\}$ is not correct provenance for o with respect to W. To show that $I_1^* - \{e_1\}$ is not correct, we will construct an input subset $I_1' \subseteq I_1^* \subseteq I_1$ such that $o \in W(I_1')$ and $o \notin W(I_1' - \{e_1\})$.

Let e_1 be any element in I_1^*. $I_{k+1}^* = P_{T_{k+1}}(o)$ is the minimal provenance of o with respect to T_{k+1}. Since $I_1^* = \bigcup_{e \in I_{k+1}^*} P_W(e)$, we know that $e_1 \in P_W(e_{k+1})$ for some $e_{k+1} \in I_{k+1}^*$. By Lemma 3.1, there exists a minimal witness $I_{k+1}^w \subseteq I_{k+1}^*$ of o with respect to T_{k+1} that contains e_{k+1}. Consider $P_W(e_{k+1}) \subseteq I_1$, the workflow provenance of $e_{k+1} \in I_{k+1}^w$ in W. By the inductive hypothesis, $P_W(e_{k+1})$ is minimal for e_{k+1} with respect to $W_k = T_1 \circ \ldots \circ T_k$. By Lemma 3.1, since $P_W(e_{k+1})$ is minimal for e_{k+1} and $e_1 \in P_W(e_{k+1})$, there exists a minimal witness $I_1^w \subseteq P_W(e_{k+1}) \subseteq I_1$ for e_{k+1} with respect to W_{k+1} that contains e_1.

Let $P(e)$ be the minimal provenance of $e \in I_{k+1}$ with respect to W_k. We construct subset $I_1' \subseteq I_1$ in the following way: $I_1' = \bigcup_{e \in I_{k+1}^w} D(e)$ where $D(e) = P(e)$ if $e \neq e_{k+1}$, and $D(e_{k+1}) = I_1^w \subseteq P(e_{k+1})$.

Since I_1' contains a minimal witness for every element $e \in I_{k+1}^w$ with respect to W_k, we know that $W_k(I_1') \supseteq I_{k+1}^w$, implying that $W(I_1') = T_{k+1}(W_k(I_1')) \supseteq T_{k+1}(I_{k+1}^w)$ since T_{k+1} is monotonic. $T_{k+1}(I_{k+1}^w) \supseteq \{o\}$, from which it follows that $o \in W(I_1')$.

Consider $W(I_1' - \{e_1\}) = T_{k+1}(W_k(I_1' - \{e_1\}))$. For all $e \in (I_{k+1}^w - \{e_{k+1}\})$, since W_k is m-1 (by Lemma 3.2), and $D(e)$ is minimal (and thus correct) for e with respect to W_k, we know that $(W_k(I_1' - \{e_1\}) \cap \{e\}) = (W_k((I_1' - \{e_1\}) \cap D(e)) \cap \{e\}) = (W_k(D(e)) \cap \{e\}) = \{e\}$, implying that $e \in (W_k(I_1' - \{e_1\})$. For $e \notin I_{k+1}^w$, $(W_k(I_1' - \{e_1\}) \cap \{e\}) = (W_k((I_1' - \{e_1\}) \cap P(e)) \cap \{e\}) = W_k(\emptyset) \cap \{e\} = \emptyset$, implying that $e \notin W_k(I_1' - \{e_1\})$. For element e_{k+1}, since W_k is m-1, and $P(e_{k+1})$ is minimal (and thus correct) for e_{k+1} with respect to W_k, $(W_k(I_1' - \{e_1\}) \cap \{e_{k+1}\}) = W_k((I_1' - \{e_1\}) \cap P(e_{k+1})) \cap \{e_{k+1}\} = W_k(I_1^w - \{e_1\}) \cap \{e\} = \emptyset$, implying that $e_{k+1} \notin W_k(I_1' - \{e_1\})$. Thus, $W_k(I_1' - \{e_1\}) = I_{k+1}^w - \{e_{k+1}\}$. It follows that $W(I_1' - \{e\}) = T_{k+1}(I_{k+1}^w - \{e_{k+1}\}) = \emptyset$, proving that $I_1^* - \{e_1\}$ is not correct provenance for o with respect to W, and thus I_1^* is minimal for o with respect to W_{k+1}. □

Proof of Theorem 3.2 For each data set I_i in the workflow instance, define $I_i^* \subseteq I_i$ recursively:

- $I_{n+1}^* = \{o\}$
- For $i \leq n$, $I_i^* = \bigcup_{e \in I_{i+1}^*} P_{T_i}(e)$

It is straightforward to see that $I_1^* = P_W(o) \subseteq I_1$. For any 1-m transformation T, the minimal provenance $P_T(e)$ of any element e in T's output has size 1. Since $T_{j+1}, \ldots, T_n$ are 1-m, we know that $|I_{j+1}^*| = 1$. Let $I_{j+1}^* = \{e_{j+1}\}$.

By Lemma 3.3, $P_W(e_{j+1}) = I_1^*$ is minimal for e_{j+1} with respect to $W_j = T_1 \circ \ldots \circ T_j$. To show that workflow provenance I_1^* is minimal for o with respect to W, let e_1 be any element in I_1^*. We need to show that $I_1^* - \{e_1\}$ is not correct provenance for o with respect to W. We construct an input subset $I_1' \subseteq I_1^* \subseteq I_1$ such that $o \in W(I_1')$ and $o \notin W(I_1' - \{e_1\})$. Since $I_1^* = P_W(e_{j+1})$ is minimal provenance for e_{j+1} with respect to W_j, by Lemma 3.1 there exists a minimal witness $I^w \subseteq I_1^*$ of e_{j+1} with respect to W_j containing e_1.

Let $I_1' = I^w$. Then $e_{j+1} \notin (T_1 \circ \ldots \circ T_j)(I_1' - \{e_1\})$, implying that $o \notin W(I_1' - \{e_1\})$. However, $e_{j+1} \in (T_1 \circ \ldots \circ T_j)(I_1')$, implying that $o \in W(I_1')$. Thus, workflow provenance is minimal. □

3.4 Preservation of Weak Correctness

Given a workflow with at most one nonmonotonic transformation, if we have weakly correct provenance at each transformation, then workflow provenance is guaranteed to be weakly correct.

Theorem 3.3 Let $W = T_1 \circ T_2 \circ \ldots \circ T_n$. Consider a workflow instance $(T_1 \circ T_2 \circ \ldots \circ T_n)(I_1) = I_{n+1}$ in which at most one of the transformations is nonmonotonic. For each T_i and each element $e \in I_{i+1}$, let $P_{T_i}(e) \subseteq I_i$ be weakly correct provenance for e with respect to T_i. Consider any output element $o \in I_{n+1}$. Workflow provenance $P_W(o)$ is weakly correct for o with respect to W.

Proof. For each data set I_i in the workflow instance, define $I_i^* \subseteq I_i$ recursively:

- $I_{n+1}^* = \{o\}$
- For $i \leq n$, $I_i^* = \bigcup_{e \in I_{i+1}^*} P_{T_i}(e)$

It is straightforward to see that $I_1^* = P_W(o) \subseteq I_1$. Given an input subset $I_1' \subseteq I_1$ define sets F_i as:

- $F_1 = I_1'$
- For $2 \leq i \leq n+1$, $F_i = T_{i-1}(F_{i-1})$

Note that $F_{n+1} = W(I_1')$. Recall from Definition 3.5 that weak correctness requires $o \in T(I')$ for every $I^* \subseteq I' \subseteq I$. If every transformation T_i is monotonic and $I_1' \supseteq I_1^*$, then $F_i \supseteq I_i^*$ for all i. It follows that $W(I_1') = F_{n+1} \supseteq I_{n+1}^* = \{o\}$, implying $o \in W(I_1')$ for any input subset I_1' such that $I_1^* \subseteq I_1' \subseteq I_1$, and thus workflow provenance I_1^* is weakly correct.

Suppose there is one nonmonotonic transformation T_j. Based on the above argument, since the workflow before T_j is monotonic, $I_j^* \subseteq F_j \subseteq I_j$. For each element $e \in I_{j+1}^*$, since $P_{T_j}(e) \subseteq I_j^*$, we know that $P_{T_j}(e) \subseteq F_j \subseteq I_j$, and since $P_{T_j}(e)$ is weakly correct, we know that $e \in T_j(F_j) = F_{j+1}$. Thus $F_{j+1} \supseteq I_{j+1}^*$. Since the transformations after T_j are all monotonic and $F_{j+1} \supseteq I_{j+1}^*$, we know that $F_i \supseteq I_i^*$ for all $i > j+1$. Thus, $W(I_1') = F_{n+1} \supseteq I_{n+1}^* = \{o\}$ for any input subset I_1' such that $I_1^* \subseteq I_1' \subseteq I_1$. Hence workflow provenance I_1^* is weakly correct. □

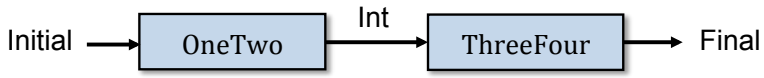

Figure 5: Counterexample for weakly correct workflow provenance.

Given a workflow with two nonmonotonic transformations, workflow provenance is not guaranteed to be correct or weak correct.

Example 3.3 Consider workflow $W = \mathbf{OneTwo} \circ \mathbf{ThreeFour}$ shown in Figure 5. Let $\mathsf{Initial} = \{1, 2\}$. Transformation **OneTwo** produces intermediate data set Int as follows:

- If $1 \in \mathsf{Initial}$ and $2 \in \mathsf{Initial}$, then $\mathbf{OneTwo}(\mathsf{Initial}) = \{3\}$
- Else if $1 \in \mathsf{Initial}$ and $2 \notin \mathsf{Initial}$, then $\mathbf{OneTwo}(\mathsf{Initial}) = \{3, 4\}$
- Else, $\mathbf{OneTwo}(\mathsf{Initial}) = \emptyset$

In this workflow instance, $\mathsf{Int} = \{3\}$. The following is correct (and therefore weakly correct) provenance for 3 with respect to **OneTwo**: $P_{\mathbf{OneTwo}}(3) = \{1\}$. (Intuitively $\{1\}$ is correct, and in fact minimal, because any time $\mathsf{Initial}$ contains 1, Int contains 3.) Transformation **ThreeFour** produces Final from Int as follows:

- If $3 \in \mathsf{Int}$ and $4 \notin \mathsf{Int}$, then $\mathbf{ThreeFour}(\mathsf{Int}) = \{5\}$
- Else, $\mathbf{ThreeFour}(\mathsf{Int}) = \emptyset$

In this workflow instance, $\mathsf{Final} = \{5\}$. The following provenance is correct (and therefore weakly correct) for 5 with respect to **ThreeFour**: $P_{\mathbf{ThreeFour}}(5) = \{3\}$. Neither **OneTwo** nor **ThreeFour** is monotonic. Let $o = 5 \in \mathsf{Final}$. The workflow provenance $P_W(o)$ of o is $\{1\}$. However, the only weakly correct provenance of o with respect to W is $\{1, 2\}$, since $5 \notin W(\{1\}) = \emptyset$. Thus, workflow provenance is not weakly correct. □

3.5 Workflows with Multi-Input and Multi-Output Transformations

We now give a generalized definition of workflow provenance that supports workflows containing multi-input and multi-output transformations, i.e., the workflow is a directed graph, but without cycles.

Definition 3.7 (Workflow Provenance) Consider a workflow instance $W(I_1, \ldots, I_m)$ with initial inputs $I_1, \ldots, I_m$. Let e be any data element involved in the workflow—input, intermediate, or output. The *workflow provenance* of e in W, denoted $P_W(e)$, is an m-tuple $(I_1^*, \ldots, I_m^*)$, where $I_1^* \subseteq I_1, \ldots, I_m^* \subseteq I_m$. If e is an initial input element, i.e., $e \in I_k$, then $P_W(e) = (\emptyset, \ldots, \{e\}, \ldots, \emptyset)$, a tuple in which every value except the k-th is empty. Otherwise, let e be an element in one of the output sets of a transformation T. Let $P_T(e) = (P_1^T(e), \ldots, P_n^T(e))$ be the one-level provenance of e with respect to T. Then $P_W(e) = (P_1^W(e), \ldots, P_m^W(e))$ where each $P_i^W(e) = \bigcup_{e' \in (P_1^T(e) \cup \ldots \cup P_n^T(e))} P_i^W(e')$. □

In this more general setting, we again want to determine when the properties of correctness, minimality, and weak correctness carry over from the individual transformations' provenance to the workflow provenance. We first generalize the definitions of *monotonicity* and *weak correctness*.

Definition 3.8 (Monotonicity) A transformation T is *monotonic* if:

- For any input sets $I_1, \ldots, I_m$ and $I'_1, \ldots, I'_m$ with $I_i \subseteq I'_i$, let $(O_1, \ldots, O_r) = T(I_1, \ldots, I_m)$ and $(O'_1, \ldots, O'_r) = T(I'_1, \ldots, I'_m)$. $O_1 \subseteq O'_1, \ldots, O_r \subseteq O'_r$. □

Definition 3.9 (Weak Correctness Definition) Let $(O_1, \ldots, O_r) = T(I_1, \ldots, I_m)$ be a transformation instance and let $o \in O_i$ for some i. We say that provenance $\langle I^*_1, \ldots, I^*_m \rangle$, $I^*_1 \subseteq I_1, \ldots, I^*_m \subseteq I_m$, is *weakly correct* for o with respect to T if:

- For any $\langle I'_1, \ldots, I'_m \rangle$ such that $I^*_1 \subseteq I'_1 \subseteq I_1, \ldots, I^*_m \subseteq I'_m \subseteq I_m$, $o \in T(I'_1, \ldots, I'_m)$. □

We now generalize the major theorems presented in this section. Proofs are omitted since they are very similar to the proofs given earlier for linear workflows; no new complexities are encountered.

Theorem 3.4 (Preservation of Correctness) Let W be a workflow containing monotonic transformations $T_1, \ldots, T_n$. Consider a workflow instance $W(I_1, \ldots, I_m)$ with initial inputs $I_1, \ldots, I_m$. For each T_i, and each element e in any of T_i's output sets, let $P_{T_i}(e)$ be correct provenance for e with respect to T_i. Consider any output element o. Workflow provenance $P_W(o)$ is correct for o with respect to W. □

Theorem 3.5 (Preservation of Minimality) Let W be a workflow containing monotonic transformations $T_1, \ldots, T_n$, in which each transformation is either m-1 or 1-m, and along no path in the workflow is there a 1-m transformation followed by an m-1 transformation. Consider a workflow instance $W(I_1, \ldots, I_m)$ with initial inputs $I_1, \ldots, I_m$. For each T_i, and each element e in any of T_i's output sets, let $P_{T_i}(e)$ be minimal provenance for e with respect to T_i. Consider any output element o. Workflow provenance $P_W(o)$ is minimal for o with respect to W. □

Theorem 3.6 (Preservation of Weak Correctness) Let W be a workflow containing transformations $T_1, \ldots, T_n$, in which each path contains at most one nonmonotonic transformation. Consider a workflow instance $W(I_1, \ldots, I_m)$ with initial inputs $I_1, \ldots, I_m$. For each T_i, and each element e in any of T_i's output sets, let $P_{T_i}(e)$ be weakly correct provenance for e with respect to T_i. Consider any output element o. Workflow provenance $P_W(o)$ is weakly correct for o with respect to W. □

Note that the workflow counterexamples (Examples 3.1, 3.2, 3.3) given earlier are also valid in this more general setting.

4 Logical Provenance Specifications

Having discussed the desirable formal properties of provenance in Section 2 and how these properties carry over from individual transformations to a workflow in Section 3, we now discuss how to actually specify provenance relationships between the input elements and output elements of a given transformation. Recall we assume each data set has some predefined attributes that are present in each element. Elements may contain arbitrary additional data, which we do not use for provenance.

We describe a simple logical-provenance (transformation level) specification language. This language can encode precise (data-element level) provenance for a large class of transformations, capturing the same information as physical provenance but in a much more compact fashion. Logical provenance specifications in our language consist of *attribute mappings* and *filters*.

Definition 4.1 (Attribute Mapping) Let T be a transformation with input sets $I_1, \ldots, I_m$ and output sets $O_1, \ldots, O_r$. Let A be an attribute of input set I_i, and let B be an attribute of output set O_j. Given any value $x \in \mathit{domain}(B)$ and any input set instances $I'_1, \ldots, I'_m$, let $\sigma_{B=x}(T(I'_1, \ldots, I'_m))$ be shorthand for $\sigma_{B=x}(O'_j)$ where $(O'_1, \ldots, O'_r) = T(I'_1, \ldots, I'_m)$. T has an *attribute mapping* between input attribute $I_i.A$ and output attribute $O_j.B$, denoted $I_i.A \leftrightarrow O_j.B$, if for all possible values of $x \in \mathit{domain}(B)$ and all possible input set instances $I'_1, \ldots, I'_m$:

$\sigma_{B=x}(T(I'_1, \ldots, I'_m)) = \sigma_{B=x}(T(I'_1, \ldots, \sigma_{A=x}(I'_i), \ldots, I'_m))$ □

In words, attribute mapping $I_i.A \leftrightarrow O_j.B$ states that the output subset of O_j where $B = x$ is unaffected by all elements in I_i except those where $A = x$.

Example 4.1 Consider transformation **JoinAgg** from the running example (Figure 1), which takes data sets CustSales (CS) and ItemProfit (IP) as input and joins them on attribute item-id to output data set ItemCountryProfit (IC). Attribute mapping IP.brand ↔ IC.brand is one example that holds for this transformation, indicating that the subset of output elements in IC with particular brand values is produced by the input subset of IP with the same brand values. Other attribute mappings that hold for transformation **JoinAgg** are IP.item-id ↔ IC.item-id, IP.type ↔ IC.type, CS.country ↔ IC.country, and CS.item-id ↔ IC.item-id. □

Definition 4.2 (Filter) Let T be a transformation with input sets $I_1, \ldots, I_m$ and output sets $O_1, \ldots, O_r$. T has a *filter condition* C on input I_i if for all possible input set instances $I'_1, \ldots, I'_m$:

$T(I'_1, \ldots, I'_m) = T(I'_1, \ldots, \sigma_C(I'_i), \ldots, I'_m)$ □

In words, filter C states that output elements are never affected by input elements from I_i that do not satisfy condition C.

Example 4.2 Consider transformation **Filter** from the running example, which takes data set ItemCountryProfit (IC) as input and outputs data set LaptopProfit (LP). The filter type='laptop' holds for this transformation, indicating that output elements in LP are never affected by elements in IC except those satisfying type='laptop'. □

We now define provenance as encoded by attribute mappings and filters. We will shortly prove its correctness.

Definition 4.3 (Provenance Encoded by Logical Specification) Consider a transformation instance $(O_1, \ldots, O_r) = T(I_1, \ldots, I_m)$ with a logical provenance specification consisting of a set of attribute mappings M and a set of filters F. We denote the specification as (M, F). Let $M_{i,j}$ denote the subset of attribute mappings $A \leftrightarrow B$ in M such that attribute A is from input I_i and attribute B is from output O_j. Let F_i denote the subset of filters in F that are on I_i. Consider output element $o \in O_j$ for some j. The provenance of o as encoded by (M, F) is $\langle I_1^*, \ldots, I_m^* \rangle = \langle \sigma_{C_1}(I_1), \ldots, \sigma_{C_m}(I_m) \rangle$ where each $C_i = (\bigwedge_{(A \leftrightarrow B) \in M_{i,j}} A = o.B) \wedge (\bigwedge_{C \in F_i} C)$. □

Example 4.3 Consider again transformation **Filter** from the running example, which takes data set ItemCountryProfit (IC) as input and outputs data set LaptopProfit (LP). Consider the logical provenance specification (M, F), where $M = \{$IC.item-id↔LP.item-id, IC.country ↔ LP.country, IC.brand ↔ LP.brand$\}$ and $F = \{$type='laptop'$\}$. Referring to Figure 2, if $o =$ LP(1), the first tuple in LP, then o's provenance as encoded by (M, F) is $\sigma_{\text{item-id='I1'} \wedge \text{country='France'} \wedge \text{brand='HP'} \wedge \text{type='laptop'}}(\text{IC}) = \{\text{IC}(1)\}$. □

We now prove that provenance encoded by logical specifications is indeed correct, according to Definition 2.4 of provenance correctness.

Theorem 4.1 Consider a transformation instance $(O_1, \ldots, O_r) = T(I_1, \ldots, I_m)$ with logical provenance specification (M, F). Given any output element $o \in O_j$ for some j, o's provenance $\langle I_1^*, \ldots I_m^* \rangle = \langle \sigma_{C_1}(I_1), \ldots, \sigma_{C_m}(I_m) \rangle$ as specified in Definition 4.3 is correct for o with respect to T according to Definition 2.4.

Proof. Consider any output element $o \in O_j$ for some j. $\langle I_1^*, \ldots, I_m^* \rangle = \langle \sigma_{C_1}(I_1), \ldots, \sigma_{C_m}(I_m) \rangle$ where each C_i is a conjunction of predicates as in Definition 4.3. Consider any $\langle I_1', \ldots, I_m' \rangle$ such that $I_1' \subseteq I_1, \ldots, I_m' \subseteq I_m$. We need to show that $o \in T(I_1', \ldots, I_m')$ if and only if $o \in T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*)$.

First suppose $o \in T(I_1', \ldots, I_m')$. Then $o \in \sigma_{\mathbf{B}=o.\mathbf{B}}(T(I_1', \ldots, I_m'))$, where $\mathbf{B}$ contains all attributes in o. Since M and F hold for T, $\sigma_{\mathbf{B}=o.\mathbf{B}}(T(I_1', \ldots, I_m')) = \sigma_{\mathbf{B}=o.\mathbf{B}}(T(\sigma_{C_1}(I_1'), \ldots, \sigma_{C_m}(I_m')))$, from which it follows that $o \in \sigma_{\mathbf{B}=o.\mathbf{B}}(T(\sigma_{C_1}(I_1'), \ldots, \sigma_{C_m}(I_m')))$. Since $T(\sigma_{C_1}(I_1'), \ldots, \sigma_{C_m}(I_m')) = T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*)$, $o \in \sigma_{\mathbf{B}=o.\mathbf{B}}(T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*))$, implying $o \in T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*)$.

Now suppose $o \in T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*)$. Since $T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*) = T(\sigma_{C_1}(I_1'), \ldots, \sigma_{C_m}(I_m'))$, $o \in T(\sigma_{C_1}(I_1'), \ldots, \sigma_{C_m}(I_m'))$. Then $o \in \sigma_{\mathbf{B}=o.\mathbf{B}}(T(\sigma_{C_1}(I_1'), \ldots, \sigma_{C_m}(I_m')))$. Since M and F hold for T,

$\sigma_{\mathbf{B}=o.\mathbf{B}}(T(\sigma_{C_1}(I'_1),\ldots,\sigma_{C_m}(I'_m))) = \sigma_{\mathbf{B}=o.\mathbf{B}}(T(I'_1,\ldots,I'_m))$, from which it follows that $o \in \sigma_{\mathbf{B}=o.\mathbf{B}}(T(\sigma_{C_1}(I'_1),\ldots,\sigma_{C_m}(I'_m)))$. Thus, $o \in \sigma_{\mathbf{B}=o.\mathbf{B}}(T(I'_1,\ldots,I'_m))$, implying $o \in T(I'_1,\ldots,I'_m)$. □

4.1 Attribute Mappings with Functions

We now show how attribute mappings can be extended beyond simple equalities to involve functions.

Definition 4.4 (Attribute Mapping with Functions) Let T be a transformation with input sets $I_1,\ldots,I_m$ and output sets $O_1,\ldots,O_r$. Let $\mathbf{A}$ be attributes of input set I_i, and let $\mathbf{B}$ be attributes of output set O_j. Let f be a function whose domain includes all possible values of $\mathbf{A}$, and let g be a function whose domain includes all possible values of $\mathbf{B}$. T has an attribute mapping between $f(I_i.\mathbf{A})$ and $g(O_j.\mathbf{B})$, denoted $f(I_i.\mathbf{A}) \leftrightarrow g(O_j.\mathbf{B})$, if for all possible values of $g(\mathbf{B})$ and all possible input set instances $I'_1,\ldots,I'_m$:
$\sigma_{g(\mathbf{B})=x}(T(I'_1,\ldots,I'_m)) = \sigma_{g(\mathbf{B})=x}(T(I'_1,\ldots,\sigma_{f(\mathbf{A})=x}(I'_i),\ldots,I'_m))$. □

Example 4.4 Consider a transformation **NameCombine** that takes input set Emp(first-name, last-name, addr) and creates output set Person(name, addr) by concatenating first-name and last-name for each record in Emp. Let $f(a,b)$ be a function that concatenates a and b. Attribute mapping f(Emp.first-name, Emp.last-name) $\leftrightarrow$ Person.name holds for this transformation, indicating that the subset of output elements in Person with particular name values corresponds to the input subset of Emp for which concatenating first-name and last-name produces that name. □

In the remainder of this paper, we will only consider mappings that involve equalities. Many of the theorems that follow can be extended to handle functions without introducing any new complexities.

4.2 Multi-Attribute Mappings

Given attribute mappings involving single input and output attributes, we can combine them to create attribute mappings across sets of attributes.

Definition 4.5 (Multi-Attribute Mapping) Let T be a transformation with input sets $I_1,\ldots,I_m$ and output sets $O_1,\ldots,O_r$. Let $\mathbf{A} = (A_1, A_2, \ldots, A_n)$ and $\mathbf{B} = (B_1, B_2, \ldots, B_n)$ denote vectors of attributes from $\mathbf{I} = (I_1,\ldots,I_m)$ and a single output set O_j respectively. (We explain below why multiple output sets are not considered.) Let $\sigma_{\mathbf{A}=\mathbf{x}}(I_1,\ldots,I_m)$ denote $(\sigma_{p_1}(I_1),\ldots,\sigma_{p_m}(I_m))$ where p_i is the conjunction of all terms in the set $\{A_1 = x_1,\ldots,A_n = x_n\}$ such that A_j is an attribute in I_i. Given any $\mathbf{x} \in \mathit{domain}(\mathbf{B})$ and any input set instances $I'_1,\ldots,I'_m$, let $\sigma_{\mathbf{B}=\mathbf{x}}(T(I'_1,\ldots,I'_m))$ be shorthand for $\sigma_{B_1=x_1\wedge\ldots\wedge B_n=x_n}(O'_j)$ where $(O'_1,\ldots,O'_r) = T(I'_1,\ldots,I'_m)$. Attribute mapping $\mathbf{I}.\mathbf{A} \leftrightarrow O_j.\mathbf{B}$ holds if $\forall \mathbf{I}, \mathbf{x} \in \mathit{domain}(\mathbf{B})$, we have $\sigma_{\mathbf{B}=\mathbf{x}}(T(I_1,\ldots,I_m)) = \sigma_{\mathbf{B}=\mathbf{x}}(T(\sigma_{\mathbf{A}=\mathbf{x}}(I_1,\ldots,I_m)))$.

Example 4.5 Consider again transformation **JoinAgg** from the running example. Attribute mapping (IP.item-id, IP.brand, IP.type) $\leftrightarrow$ (IC.item-id, IC.brand, IC.type) holds for this transformation, indicating that the subset of output elements in IC with particular (item-id, brand, type) values corresponds to the input subset of IP with the same (item-id, brand, type) values. □

The following theorem states that a set of single-attribute mappings is equivalent to its combination:

Theorem 4.2 (Combining Attribute Mappings) Let T be a transformation with input sets $I_1, \ldots, I_m$ and output sets $O_1, \ldots, O_r$. Let $\mathbf{A} = (A_1, A_2, \ldots, A_n)$ and $\mathbf{B} = (B_1, B_2, \ldots, B_n)$ denote vectors of attributes from $\mathbf{I} = (I_1, \ldots, I_m)$ and a single output set O_j respectively. Attribute mappings $A_1 \leftrightarrow B_1, \ldots, A_n \leftrightarrow B_n$ hold if and only if $\mathbf{A} \leftrightarrow \mathbf{B}$ holds.

Proof. We prove by induction on the number of attributes. For $n = 1$, the theorem follows from the assumption. Now suppose that the theorem holds for $n = k$ and consider $n = k + 1$.

Suppose $A_1 \leftrightarrow B_1, \ldots, A_{k+1} \leftrightarrow B_{k+1}$ all hold. Since the theorem holds for $n = k$, we know that $(A_1, A_2, \ldots, A_k) \leftrightarrow (B_1, B_2, \ldots, B_k)$ holds, from which it follows that $\sigma_{\mathbf{B_k}=\mathbf{x_k}}(T(I_1, \ldots, I_m)) = \sigma_{\mathbf{B_k}=\mathbf{x_k}}(T(\sigma_{\mathbf{A_k}=\mathbf{x_k}}(I_1, \ldots, I_m)))$ for all possible values of $\mathbf{x_k}$. We can then use attribute mapping $A_{k+1} \leftrightarrow B_{k+1}$ to deduce that for all possible values of $\mathbf{x_{k+1}}$, we have that

$\sigma_{\mathbf{B_{k+1}}=\mathbf{x_{k+1}}}(T(I_1, \ldots, I_m))$
$= \sigma_{(\mathbf{B_k}=\mathbf{x_k})\wedge(B_{k+1}=x_{k+1})}(T(I_1, \ldots, I_m))$
$= \sigma_{\mathbf{B_k}=\mathbf{x_k}}(\sigma_{B_{k+1}=x_{k+1}}(T(I_1, \ldots, I_m)))$
$= \sigma_{\mathbf{B_k}=\mathbf{x_k}}(\sigma_{B_{k+1}=x_{k+1}}(T(\sigma_{A_{k+1}=x_{k+1}}(I_1, \ldots, I_m))))$ (via $A_{k+1} \leftrightarrow B_{k+1}$)
$= \sigma_{B_{k+1}=x_{k+1}}(\sigma_{\mathbf{B_k}=\mathbf{x_k}}(T(\sigma_{A_{k+1}=x_{k+1}}(I_1, \ldots, I_m))))$
$= \sigma_{B_{k+1}=x_{k+1}}(\sigma_{\mathbf{B_k}=\mathbf{x_k}}(T(\sigma_{\mathbf{A_k}=\mathbf{x_k}}(\sigma_{A_{k+1}=x_{k+1}}(I_1, \ldots, I_m)))))$ (via assumption that theorem holds for $n = k$)
$= \sigma_{\mathbf{B_{k+1}}=\mathbf{x_{k+1}}}(T(\sigma_{\mathbf{A_{k+1}}=\mathbf{x_{k+1}}}(I_1, \ldots, I_m)))$.

Thus, if $A_1 \leftrightarrow B_1, \ldots, A_{k+1} \leftrightarrow B_{k+1}$, then $(A_1, A_2, \ldots, A_{k+1}) \leftrightarrow (B_1, B_2, \ldots, B_{k+1})$.

Now suppose $(A_1, A_2, \ldots, A_{k+1}) \leftrightarrow (B_1, B_2, \ldots, B_{k+1})$. We need to show that $A_j \leftrightarrow B_j$ holds for all j. Without loss of generality, we will set $j = k + 1$ to simplify notation. Suppose $A_{k+1} \leftrightarrow B_{k+1}$ does not hold. Then there exists x_{k+1}, $I'_1, \ldots, I'_m$ such that:

$\sigma_{B_{k+1}=x_{k+1}}(T(I'_1, \ldots, I'_m)) \neq \sigma_{B_{k+1}=x_{k+1}}(T(\sigma_{A_{k+1}=x_{k+1}}(I'_1, \ldots, I'_m)))$

Given this inequality, there exists at least one element e that is in the symmetric difference of the two sides. Let $\mathbf{x_k}$ be the value of $e.\mathbf{B_k}$. Then we have:

$\sigma_{(\mathbf{B_k}=\mathbf{x_k})\wedge(B_{k+1}=x_{k+1})}(T(I'_1, \ldots, I'_m)) \neq \sigma_{(\mathbf{B_k}=\mathbf{x_k})\wedge(B_{k+1}=x_{k+1})}(T(\sigma_{A_{k+1}=x_{k+1}}(I'_1, \ldots, I'_m)))$
$\sigma_{\mathbf{B_{k+1}}=\mathbf{x_{k+1}}}(T(I'_1, \ldots, I'_m)) \neq \sigma_{\mathbf{B_{k+1}}=\mathbf{x_{k+1}}}(T(\sigma_{A_{k+1}=x_{k+1}}(I'_1, \ldots, I'_m)))$

By the definition of $(A_1, A_2, \ldots, A_{k+1}) \leftrightarrow (B_1, B_2, \ldots, B_{k+1})$ holding for T, we know that for all $I_1, \ldots, I_m$:

$\sigma_{\mathbf{B_{k+1}}=\mathbf{x_{k+1}}}(T(I_1, \ldots, I_m)) = \sigma_{\mathbf{B_{k+1}}=\mathbf{x_{k+1}}}(T(\sigma_{\mathbf{A_{k+1}}=\mathbf{x_{k+1}}}(I_1, \ldots, I_m)))$

Setting $(I_1, \ldots, I_m)$ equal to $(I'_1, \ldots, I'_m)$ we get:

$\sigma_{\mathbf{B_{k+1}=x_{k+1}}}(T(I'_1,\ldots,I'_m)) = \sigma_{\mathbf{B_{k+1}=x_{k+1}}}(T(\sigma_{\mathbf{A_{k+1}=x_{k+1}}}(I'_1,\ldots,I'_m)))$
Setting $(I_1,\ldots,I_m)$ equal to $\sigma_{A_{k+1}=x_{k+1}}(I'_1,\ldots,I'_m)$ we get:
$\sigma_{\mathbf{B_{k+1}=x_{k+1}}}(T(\sigma_{A_{k+1}=x_{k+1}}(I'_1,\ldots,I'_m))) = \sigma_{\mathbf{B_{k+1}=x_{k+1}}}(T(\sigma_{\mathbf{A_{k+1}=x_{k+1}}}(I'_1,\ldots,I'_m)))$
But then we have $\sigma_{\mathbf{B_{k+1}=x_{k+1}}}(T(I'_1,\ldots,I'_m)) = \sigma_{\mathbf{B_{k+1}=x_{k+1}}}(T(\sigma_{A_{k+1}=x_{k+1}}(I'_1,\ldots,I'_m)))$, a contradiction. Thus, $A_{k+1} \leftrightarrow B_{k+1}$ holds. □

In Definition 4.5 and Theorem 4.2 we have not considered multi-attribute mappings involving multiple output sets. While we could extend Definition 4.5 in a natural way to allow multiple output sets, Theorem 4.2 would not hold, as demonstrated by the following example.

Example 4.6 Consider transformation T with input data sets $I_1(A) = \{(1),(2),(3)\}$ and $I_2(B) = \{(2),(3),(4)\}$, and output data sets $O_1(C) = \{(2),(3)\}$ and $O_2(D) = \{(2),(3)\}$. Both O_1 and O_2 are computed by taking the intersection of the values in I_1 and I_2. Attribute mappings $I_1.A \leftrightarrow O_1.C$ and $I_2.B \leftrightarrow O_2.D$ both hold individually. However, consider multi-attribute mapping $(I_1.A, I_2.B) \leftrightarrow (O_1.C, O_2.D)$ defined as follows: For any $x \in$ *domain*(C), $y \in$ *domain*(D) and any input set instances I'_1, I'_2, we have $\sigma_{C=x}(T(I'_1,I'_2)) = \sigma_{C=x}(T(\sigma_{A=x}(I'_1),\sigma_{B=y}(I'_2)))$ and $\sigma_{D=y}(T(I'_1,I'_2)) = \sigma_{D=y}(T(\sigma_{A=x}(I'_1),\sigma_{B=y}(I'_2)))$. Recall $(O_1,O_2) = T(I_1,I_2)$. Let $(O'_1,O'_2) = T(\sigma_{A=2}(I_1),\sigma_{B=3}(I_2))$. Note that $O'_1 = \emptyset$ and $O'_2 = \emptyset$. Mapping $(I_1.A, I_2.B) \leftrightarrow (O_1.C, O_2.D)$ implies that $\sigma_{C=2}(O_1) = \sigma_{C=2}(O'_1)$ and $\sigma_{D=3}(O_2) = \sigma_{D=3}(O'_2)$. However, $\sigma_{C=2}(O_1) = \{(2)\} \neq \emptyset = \sigma_{C=2}(O'_1)$ and $\sigma_{D=3}(O_2) = \{(3)\} \neq \emptyset = \sigma_{D=3}(O'_2)$, and thus $(I_1.A, I_2.B) \leftrightarrow (O_1.C, O_2.D)$ does not hold. □

In the remainder of the paper we consider single-attribute mappings only, since multi-attribute mappings generally don't add useful expressive power.

5 Provenance Tracing in Workflows

Consider a workflow in which each transformation has a logical provenance specification in the language described in Section 4. Given an output element in the workflow, suppose we want to find the input subsets that contributed to the output element, e.g., for debugging or drill-down purposes. In this section, we discuss how to perform *provenance tracing*, i.e., how to use the logical provenance specifications given for each transformation in the workflow to compute the workflow provenance of a given output element. In particular, we are interested in making provenance tracing as efficient as possible.

As we saw in Section 3, even if we have minimal provenance for each transformation, workflow provenance isn't always minimal, or even correct. Using the theoretical results of Section 3, for many workflows we can guarantee the minimality or correctness of workflow provenance. For workflows where minimality or correctness of workflow provenance is not theoretically guaranteed, we would still like to provide the option of computing workflow provenance, since it could be helpful for debugging and drill-down.

Although it is straightforward to trace provenance recursively backwards through a workflow one transformation at a time, sometimes we can combine logical provenance across

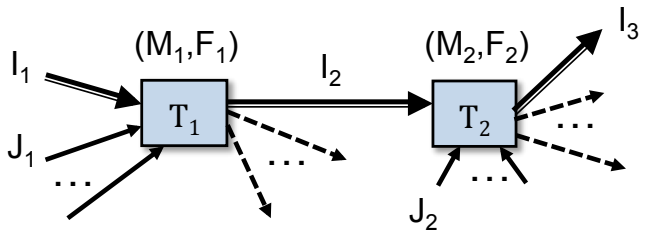

Figure 6: Abstract workflow $W = T_1 \circ T_2$.

transformations, enabling more efficient tracing. In Section 5.1, we specify when and how logical provenance can be combined across transformations without losing correctness. In Section 5.2, we give conditions under which tracing the combined logical provenance is guaranteed to return the same result as tracing each transformation's logical specification separately (thus, e.g., preserving minimality). In Section 5.3, we present our overall algorithm for provenance tracing in workflows with logical provenance specifications given for each transformation.

5.1 Combining Logical Provenance Across Transformations

Let T_1 and T_2 be transformations such that T_2 takes as input one of T_1's output data sets. Given an element o from an output set I_3 of T_2, suppose we wanted to trace o's provenance to all input sets of T_1 using the logical provenance specifications for T_1 and T_2. We specify when and how logical provenance can be combined across T_1 and T_2 without losing correctness.

Consider Figure 6. Let I_2 be the output set of T_1 that is passed to T_2. Without loss of generality, we assume T_1 has only one input set in addition to I_1, which we call J_1. Similarly T_2 has input data sets I_2 and J_2. Throughout this section, we use $W = T_1 \circ T_2$ to denote the workflow composed of T_1 and T_2, with input sets I_1, J_1, J_2. $T_1(I_1, J_1)$ may output multiple sets; as convention we use $T_1(I_1, J_1)_2$ to denote the output set of $T_1(I_1, J_1)$ that is passed to T_2. Let M_1 denote the set of attribute mappings for T_1 between I_1 and I_2, and let F_1 denote the set of filters for T_1 on I_1. Similarly let M_2 denote the set of attribute mappings for T_2 between I_2 and I_3, and let F_2 denote the set of filters for T_2 on I_2. We first show how attribute mappings can be combined across transformations.

Theorem 5.1 (Transitivity of Attribute Mappings) Consider $W = T_1 \circ T_2$. If T_1 has attribute mapping $(I_1.A \leftrightarrow I_2.B) \in M_1$, and T_2 has attribute mapping $(I_2.B \leftrightarrow I_3.C) \in M_2$, then attribute mapping $(I_1.A \leftrightarrow I_3.C) \in M_W$ holds for W.

Proof 5.1 Since $I_1.A \leftrightarrow I_2.B$ holds for T_1, we know that for all possible values of x and I_1', J_1', $\sigma_{B=x}(T_1(I_1', J_1')) = \sigma_{B=x}(T_1(\sigma_{A=x}(I_1'), J_1')))$. Since $I_2.B \leftrightarrow I_3.C$ holds for T_2, we know that for all x and I_2', J_2', $\sigma_{C=x}(T_2(I_2', J_2')) = \sigma_{C=x}(T_2(\sigma_{B=x}(I_2'), J_2'))$. Thus, we know that for all x and I_1', J_1', J_2':
$\sigma_{C=x}((T_1 \circ T_2)(I_1', J_1', J_2'))$

$= \sigma_{C=x}(T_2(T_1(I_1', J_1')_2, J_2'))$
$= \sigma_{C=x}(T_2(\sigma_{B=x}(T_1(I_1', J_1')), J_2'))$ (since $I_2.B \leftrightarrow I_3.C$ holds for T_2)
$= \sigma_{C=x}(T_2(\sigma_{B=x}(T_1(\sigma_{A=x}(I_1'), J_1')), J_2'))$ (since $I_1.A \leftrightarrow I_2.B$ holds for T_1)
$= \sigma_{C=x}(T_2(T_1(\sigma_{A=x}(I_1'), J_1')_2, J_2'))$ (since $I_2.B \leftrightarrow I_3.C$ holds for T_2)
$= \sigma_{C=x}((T_1 \circ T_2)(\sigma_{A=x}(I_1'), J_1', J_2'))$
proving that $I_1.A \leftrightarrow I_3.C$ holds for $W = T_1 \circ T_2$. □

Example 5.1 Consider transformations **JoinAgg** and **Filter** from the running example (Figure 1). Since attribute mapping IP.brand ↔ IC.brand holds for **JoinAgg**, and attribute mapping IC.brand ↔ LP.brand holds for **Filter**, then by Theorem 5.1 attribute mapping IP.brand ↔ LP.brand holds for **JoinAgg**∘**Filter**. Informally, the composite mapping states that the subset of output elements in LaptopProfit with a particular brand value are derived from the subset of ItemProfit with the same brand value. □

Now consider filters. It is straightforward to show that filters that hold for T_1 also hold for $T_1 \circ T_2$.

Theorem 5.2 Consider $W = T_1 \circ T_2$. If filter condition $C \in F_1$ holds for T_1, then $C \in F_W$ also holds for W.

Proof 5.2 Since T_1 has filter condition $C \in F_1$, we know that for all possible values of I_1', J_1', $T_1(I_1', J_1') = T_1(\sigma_C(I_1'), J_1')$. Thus, we know that for all I_1', J_1', J_2':
$(T_1 \circ T_2)(I_1', J_1', J_2')$
$= T_2(T_1(I_1', J_1')_2, J_2')$
$= T_2(T_1(\sigma_C(I_1'), J_1')_2, J_2')$
$= (T_1 \circ T_2)(\sigma_C(I_1'), J_1', J_2')$
proving that $C \in F_W$ holds for $W = T_1 \circ T_2$. □

The next theorem shows that in some cases filters on T_2 can be "propagated" backward through I_2 using attribute mappings so they hold on W.

Theorem 5.3 Consider $W = T_1 \circ T_2$. Suppose T_2 has a filter condition $C \in F_2$. If $(I_1.A_1 \leftrightarrow I_2.B_1, \ldots, I_1.A_s \leftrightarrow I_2.B_s) \subseteq M_1$ where $B_1, \ldots, B_s$ are all attributes that are involved in C, then filter condition $C' \in F_W$ holds for W, where C' is equal to C after replacing all B_i with A_i.

Proof 5.3 Since T_2 has a filter condition C on I_2, we know that for all I_1', J_1', J_2', $(T_1 \circ T_2)(I_1', J_1', J_2') = T_2(\sigma_C(T_1(I_1', J_1')_2), J_2')$. We need to show that for all I_1', J_1', $\sigma_C(T_1(I_1', J_1')_2) = \sigma_C(T_1(\sigma_{C'}(I_1'), J_1')_2)$. Suppose $\sigma_C(T_1(I_1', J_1')_2) \neq \sigma_C(T_1(\sigma_{C'}(I_1'), J_1')_2)$. Then there exists some o that is in one side but not the other. Since o is only in one side, it follows that $\sigma_{\mathbf{B}=o.\mathbf{B}}(\sigma_C(T_1(I_1', J_1')_2)) \neq \sigma_{\mathbf{B}=o.\mathbf{B}}(\sigma_C(T_1(\sigma_{C'}(I_1'), J_1')_2))$. But since the attribute mappings hold, we know $\sigma_{\mathbf{B}=o.\mathbf{B}}(\sigma_C(T_1(I_1', J_1')_2)) = \sigma_{\mathbf{B}=o.\mathbf{B}}(\sigma_C(T_1(\sigma_{\mathbf{A}=o.\mathbf{B}}(I_1'), J_1')_2)) = \sigma_{\mathbf{B}=o.\mathbf{B}}(\sigma_C(T_1(\sigma_{\mathbf{A}=o.\mathbf{B} \wedge C'}(I_1'), J_1')_2)) = \sigma_{\mathbf{B}=o.\mathbf{B}}(\sigma_C(T_1(\sigma_{C'}(I_1'), J_1')_2))$, a contradiction. Thus $\sigma_C(T_1(I_1', J_1')_2) = \sigma_C(T_1(\sigma_{C'}(I_1'), J_1')_2)$. □

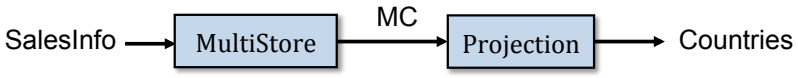

Figure 7: Example where combining logical provenance is not equivalent to workflow provenance.

5.2 Relationship Between Combined Logical Provenance and Workflow Provenance

Although attribute mappings and filters can be combined across transformations using Theorems 5.1, 5.2, and 5.3, sometimes the combined logical provenance is weaker (less precise) than the workflow provenance computed by considering transformations separately, accessing intermediate data between transformations.

Example 5.2 Consider workflow W = **MultiStore** $\circ$ **Projection** shown in Figure 7. The initial input data set SalesInfo contains country-city-sales triples for each of a corporation's worldwide stores: SalesInfo(country, city, sales) = $\{(France, Paris, 10), (France, Paris, 20), (France, Nice, 30)\}$. Transformation **MultiStore** retains cities with more than one store, producing intermediate data set MultiCities (abbreviated MC): MC(country, city) = **MultiStore**(SalesInfo) = $\{(France, Paris)\}$. Attribute mappings SalesInfo.country $\leftrightarrow$ MC.country and SalesInfo.city $\leftrightarrow$ MC.city both hold for **MultiStore**. Transformation **Projection** projects away the city name, leaving only countries (with duplicates eliminated): Countries(country) = $\{France\}$. Attribute mapping MC.country $\leftrightarrow$ Countries.country holds for **Projection**. Let $o = France \in$ Countries. The workflow provenance $P_W(o)$ of o following Definition 3.1 is $\{(France, Paris, 10), (France, Paris, 20)\} \subseteq$ SalesInfo. By Theorem 5.1, attribute mapping SalesInfo.country $\leftrightarrow$ Countries.country holds for W. However, the provenance of $o = France \in$ Countries using this composite attribute mapping is $\{(France, Paris, 10), (France, Paris, 20), (France, Nice, 30)\}$, which is correct, but not as precise as the workflow provenance $\{(France, Paris, 10), (France, Paris, 20)\}$ we computed by keeping the attribute mappings separate. □

The following theorem states that we can combine logical provenance across transformations without losing the precision of o's workflow provenance if: (1) all attribute mappings $A \leftrightarrow B$ in M_1 have a corresponding attribute mapping $B \leftrightarrow D$ in M_2, and (2) the provenance of o encoded by (M_2, F_2) for I_2 is nonempty.

Theorem 5.4 Consider workflow $W = T_1 \circ T_2$. Suppose $M_1 = \{A_1 \leftrightarrow B_1, \ldots, A_r \leftrightarrow B_r\}$ and $M_2 = \{B_1 \leftrightarrow D_1, \ldots, B_s \leftrightarrow D_s\}, r \leq s$. Let composite $M_W = \{A_1 \leftrightarrow D_1, \ldots, A_r \leftrightarrow D_r\}$. Consider a workflow instance $I_3 = W(I_1', J_1', J_2')$ and any output element $o \in I_3$. Suppose the provenance of o encoded by (M_2, F_2) for I_2 is nonempty. Let $\langle I_1^*, J_1^*, J_2^* \rangle$ be o's provenance as encoded by (M_W, F_1) for W according to Definition 4.3. Let $\langle I_1^{**}, J_1^{**}, J_2^{**} \rangle$ be the workflow provenance of o according to Definition 3.1. Then $I_1^* = I_1^{**}$.

Proof. $I_1^* = \sigma_{(A_1=o.D_1)\wedge\ldots\wedge(A_r=o.D_r)\wedge F_1}(I_1)$. We now compute I_1^{**}. The provenance of o encoded by (M_2, F_2) for I_2 is $\sigma_{(B_1=o.D_1)\wedge\ldots\wedge(B_s=o.D_s)\wedge F_2}(I_2)$. By assumption, $\sigma_{(B_1=o.D_1)\wedge\ldots\wedge(B_s=o.D_s)\wedge F_2}(I_2)$ is nonempty. Then $I_1^{**} = \bigcup_{e'\in\sigma_{(B_1=o.D_1)\wedge\ldots\wedge(B_s=o.D_s)\wedge F_2}(I_2)} \sigma_{(A_1=e'.D_1)\wedge\ldots\wedge(A_r=e'.D_r)\wedge F_1}(I_1) = \sigma_{(A_1=o.D_1)\wedge\ldots\wedge(A_r=o.D_r)\wedge F_1}(I_1) = I_1^*$. □

5.3 Tracing Algorithm

In workflows with logical specifications at each transformation, a range of provenance-tracing algorithms are possible. A conservative algorithm, for example, may combine logical provenance across transformations only when it is certain that doing so will not reduce precision. Alternatively a more aggressive algorithm (at least from the performance perspective) may combine logical provenance even without such certainty, to avoid the overhead of tracing provenance through each step of intermediate data. The tracing algorithm presented here attempts to avoid losing precision by combining logical provenance only when the primary condition of Theorem 5.4 is satisfied: Every attribute mapping $A \leftrightarrow B$ in M_1 has a corresponding mapping $B \leftrightarrow D$ in M_2. However, since the algorithm does not also check the second condition of Theorem 5.4 (whether the provenance of output elements at intermediate data sets is nonempty), in rare cases the algorithm can reduce precision. One area of future work is to study the relationship between the precision and performance of tracing algorithms, perhaps identifying classes of workflows for which tracing performance can be improved greatly without losing much precision.

Algorithm 5.1 (Provenance Tracing) Consider a workflow instance in which each transformation T has logical provenance specification (M^T, F^T). Let $I_1, \ldots, I_m$ be the initial input sets of the workflow, and let I be any intermediate or output data set. PT recursively traces the provenance of subset $E \subseteq I$ using attribute mappings M and filters F. The initial invocation of PT to trace output element $o \in O_j$ is $PT(\{o\} \subseteq O_j, \emptyset, \emptyset, O_j)$.

$PT(E \subseteq I, M, F, I')$:
  if I' is an initial input set I_j **then**:
    if $I = I'$ **then**:
      let $I_j^* = E$
      return $\langle I_1^*, \ldots, I_m^* \rangle$, where each $I_k^* = \emptyset$ for $k \neq j$
    else:
      suppose $M = \{A_1 \leftrightarrow B_1, \ldots, A_r \leftrightarrow B_r\}$
      let $I_j^* = \bigcup_{e\in E} \sigma_{(A_1=e.B_1)\wedge\ldots\wedge(A_r=e.B_r)\wedge F}(I_j)$
      return $\langle I_1^*, \ldots, I_m^* \rangle$, where each $I_k^* = \emptyset$ for $k \neq j$
  else:
    let T be the transformation that output I', with input sets $I_1^T, \ldots, I_{m_T}^T$
    for i in $[1, m_T]$:
      let M_i denote the subset of mappings $A \leftrightarrow B$ in M^T such that

A is from I_i^T and B is from I'
let F_i denote the subset of filters in F^T that are on I_i^T
if $I = I'$ **then**:
let $\langle I_1^i, \ldots, I_m^i \rangle = PT(E \subseteq I, M_i, F_i, I_i^T)$ for $1 \leq i \leq m_T$
else:
if every right attribute in M_i is a left attribute of M **then**:
suppose $M_i = \{A_1 \leftrightarrow B_1, \ldots, A_r \leftrightarrow B_r\}$
suppose $M = \{B_1 \leftrightarrow D_1, \ldots, B_s \leftrightarrow D_s\}$, $r \leq s$
let $M' = \{A_1 \leftrightarrow D_1, \ldots, A_r \leftrightarrow D_r\}$
let $\langle I_1^i, \ldots, I_m^i \rangle = PT(E \subseteq I, M', F_i, I_i^T)$
else:
suppose $M = \{A_1 \leftrightarrow B_1, \ldots, A_r \leftrightarrow B_r\}$
let $E' = \bigcup_{e \in E} \sigma_{(A_1 = e.B_1) \wedge \ldots \wedge (A_r = e.B_r) \wedge F}(I_i^T)$
let $\langle I_1^i, \ldots, I_m^i \rangle = PT(E' \subseteq I_i^T, \emptyset, \emptyset, I_i^T)$
return $\langle I_1^*, \ldots, I_m^* \rangle$, where each $I_j^* = \bigcup_{1 \leq i \leq m_T} I_j^i$

---

6 Logical Provenance for Relational Transformations

We now consider logical provenance in the relational setting. We establish notation in Section 6.1 and discuss how to generate logical provenance specifications for *Select-Project-Join* (*SPJ*) *transformations* in Section 6.2. In Section 6.3 we show that for a wide class of SPJ transformations, provenance encoded by our logical specifications is minimal. For transformations outside of this class, in Section 6.4 we introduce *augmentation*, which carries some overhead but enables minimal provenance. In Section 6.5 we extend our results to *Select-Project-Join-Aggregate* (*SPJA*) *transformations*.

6.1 Preliminaries

In the relational setting, all data sets are *tables*. A table contains a set of tuples $\{t_1, \ldots, t_n\}$ conforming to a given schema. We assume set semantics.

Definition 6.1 (Select-Project-Join Transformation) A *Select-Project-Join* (*SPJ*) *transformation* is any transformation that can be expressed as a tree of relational algebra *selection* (σ), *projection* (π), and *cross-product* ($\times$) *operators*. Note since we assume set semantics, π is duplicate-eliminating. □

All SPJ transformations T can be transformed into a canonical form $T(I_1, \ldots, I_m) = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$ using a sequence of algebraic transformations [30]. Note $\mathbf{A}$ is the final projection list, C_i are "local" (single-table) selection conditions, and C contains "join" (multi-table) conditions. Hereafter we will operate on the canonical form of relational transformations.

6.2 Logical Provenance for SPJ Transformations

Given the canonical form of an SPJ transformation T, it is straightforward to generate a set of attribute mappings and filters that hold for T.

Theorem 6.1 Let T be an SPJ transformation: $O = T(I_1, \ldots, I_m) = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$. For each attribute $I_i.A \in \mathbf{A}$, attribute mapping $I_i.A \leftrightarrow O.A$ holds for T.

Proof. Let $I'_1, \ldots, I'_m$ be any input set instances and let x be any possible value of $O.A$.
$\sigma_{A=x}(T(I'_1, \ldots, I'_m))$
$= \sigma_{A=x}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_m}(I'_m))))$
$= \sigma_{A=x}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{A=x}(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_m}(I'_m)))))$
$= \sigma_{A=x}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_i}(\sigma_{A=x}(I'_i)) \times \ldots \times \sigma_{C_m}(I'_m))))$
$= \sigma_{A=x}(T(I'_1, \ldots, \sigma_{A=x}(I'_i), \ldots, I'_m))$ □

Theorem 6.2 Let T be an SPJ transformation: $O = T(I_1, \ldots, I_m) = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$. For each I_i, T has filter condition C_i on I_i.

Proof. Let $I'_1, \ldots, I'_m$ be any input set instances. Then we have:
$T(I'_1, \ldots, I'_m)$
$= \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_i}(I'_i) \times \ldots \times \sigma_{C_m}(I'_m)))$
$= \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_i}(\sigma_{C_i}(I'_i)) \times \ldots \times \sigma_{C_m}(I'_m)))$
$= T(I'_1, \ldots, \sigma_{C_i}(I'_i), \ldots, I'_m)$ □

Given the above theorems, we can generate a *canonical logical provenance specification* for any SPJ transformation.

Definition 6.2 (Canonical Logical Specification for SPJ Transformation) Let T be an SPJ transformation: $O = T(I_1, \ldots, I_m) = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$. The *canonical logical provenance specification* for T is (M, F) where M contains all mappings $I_i.A \leftrightarrow O.A$ such that $I_i.A \in \mathbf{A}$, and F contains all C_i. □

6.3 Encoding Minimal Provenance for SPJ Transformations

We show that for a certain class of SPJ transformations, the provenance encoded by the canonical logical specification is minimal.

Theorem 6.3 Consider an SPJ transformation instance: $O = T(I_1, \ldots, I_m) = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$. Suppose $\mathbf{A}$ contains all attributes in C. (Recall C contains all multi-table conditions.) Given $o \in O$, let $\langle I^*_1, \ldots, I^*_m \rangle$ be the provenance of o as encoded by the canonical logical specification for T. $\langle I^*_1, \ldots, I^*_m \rangle$ is minimal for o with respect to T. □

The Theorem's proof follows directly from the following two Lemmas.

Lemma 6.1 Consider an SPJ transformation instance: $O = T(I_1, \ldots, I_m) = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$. Suppose that $\mathbf{A}$ contains all attributes in C. Given $o \in O$, let $\langle I_1^*, \ldots, I_m^* \rangle$ be the provenance of o as encoded by the canonical logical specification for T. Let $\mathbf{I_i}$ contain all attributes in the schema of I_i. Then $\langle I_1^*, \ldots, I_m^* \rangle = \langle I_1^{**}, \ldots, I_m^{**} \rangle$ where each $I_i^{**} = \pi_{\mathbf{I}_i}(\sigma_{C \wedge \mathbf{A}=o}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$.

Proof. Let $\mathbf{A}_i$ contain the attributes of I_i in $\mathbf{A}$. The provenance of o as encoded by T's canonical logical provenance specification is $\langle I_1^*, \ldots I_m^* \rangle = \langle \sigma_{\mathbf{A}_1 = o.\mathbf{A}_1 \wedge C_1}(I_1), \ldots, \sigma_{\mathbf{A}_m = o.\mathbf{A}_m \wedge C_m}(I_m) \rangle$. Let us show that $I_i^* = I_i^{**}$.

First suppose $e_i \in I_i^* = \sigma_{\mathbf{A}_i = o.\mathbf{A}_i \wedge C_i}(I_i)$. Then $e_i \in I_i$, and since $o \in T(I_1, \ldots, I_m) = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$, $o \in \sigma_{\mathbf{A}=o}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))) = \pi_{\mathbf{A}}(\sigma_{C \wedge (\mathbf{A}=o)}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m))) = \pi_{\mathbf{A}}(\sigma_{C \wedge (\mathbf{A}=o)}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{\mathbf{A}_i = o.\mathbf{A}_i \wedge C_i}(I_i) \times \ldots \times \sigma_{C_m}(I_m)))$. Since $\mathbf{A}$ contains all attributes in C, $\pi_{\mathbf{A}}(\sigma_{C \wedge (\mathbf{A}=o)}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{\mathbf{A}_i = o.\mathbf{A}_i \wedge C_i}(I_i) \times \ldots \times \sigma_{C_m}(I_m))) = \pi_{\mathbf{A}}(\sigma_{C \wedge (\mathbf{A}=o)}(\sigma_{C_1}(I_1) \times \ldots \times \{e_i\} \times \ldots \times \sigma_{C_m}(I_m)))$. Since $o \in \pi_{\mathbf{A}}(\sigma_{C \wedge (\mathbf{A}=o)}(\sigma_{C_1}(I_1) \times \ldots \times \{e_i\} \times \ldots \times \sigma_{C_m}(I_m)))$, there exists some $(e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_m) \in (\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_{i-1}}(I_{i-1}) \times \sigma_{C_{i+1}}(I_{i+1}) \times \ldots \times \sigma_{C_m}(I_m))$ such that $\sigma_{C \wedge \mathbf{A}=o}(\{e_1\} \times \ldots \times \{e_m\}) = \{e_1\} \times \ldots \times \{e_m\}$. Thus $I_i^{**} = \pi_{\mathbf{I}_i}(\sigma_{C \wedge \mathbf{A}=o}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m))) \supseteq \pi_{\mathbf{I}_i}(\{e_1\} \times \ldots \times \{e_m\}) \supseteq \{e_i\}$, implying $e_i \in I^{**}$.

Now suppose $e_i \in I_i^{**} = \pi_{\mathbf{I}_i}(\sigma_{C \wedge \mathbf{A}=o}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$. Then $e_i \in I_i$, $e_i.\mathbf{A}_i = o.\mathbf{A}_i$, and e_i satisfies C_i. Thus, $e_i \in \sigma_{(\mathbf{A_i}=o.\mathbf{A_i}) \wedge C_i}(I_i) = I_i^*$. □

Lemma 6.2 Consider an SPJ transformation instance: $O = T(I_1, \ldots, I_m) = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$. Given $o \in O$, consider $\langle I_1^*, \ldots, I_m^* \rangle$ where each $I_i^* = \pi_{\mathbf{I}_i}(\sigma_{C \wedge \mathbf{A}=o}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$. $\langle I_1^*, \ldots, I_m^* \rangle$ is minimal for o with respect to T.

Proof. We first show that $\langle I_1^*, \ldots, I_m^* \rangle$ is correct. Consider any $I_1' \subseteq I_1, \ldots I_m' \subseteq I_m$.

First suppose $o \in T(I_1', \ldots, I_m')$. Let us show that $o \in T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*)$. Since $o \in T(I_1', \ldots, I_m')$, there exist $e_1 \in I_1', \ldots, e_m \in I_m'$ such that $\{o\} = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(\{e_1\}) \times \ldots \times \sigma_{C_m}(\{e_m\})))$. We know $e_i \in I_i^*$, since $\{e_i\} = \pi_{\mathbf{I}_i}(\sigma_{C \wedge A=o}(\sigma_{C_1}(\{e_1\}) \times \ldots \times \sigma_{C_m}(\{e_m\}))) \subseteq \pi_{\mathbf{I}_i}(\sigma_{C \wedge A=o}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m))) = I_i^*$. Thus, $e_1 \in I_1' \cap I_1^*, \ldots, e_m \in I_m' \cap I_m^*$, and it follows that $o = T(\{e_1\}, \ldots, \{e_m\}) \subseteq T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*)$.

Now suppose $o \in T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*)$. Since T is monotonic and $I_1' \cap I_1^* \subseteq I_1', \ldots, I_m' \cap I_m^* \subseteq I_m'$, $T(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*) \subseteq T(I_1', \ldots, I_m')$, and so $o \in T(I_1', \ldots, I_m')$. Thus, $\langle I_1^*, \ldots, I_m^* \rangle$ is correct.

Let us now show $\langle I_1^*, \ldots, I_m^* \rangle$ is minimal. Suppose there existed a more precise $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ that was also correct. Then there exists some j for which $I_j^{**} \subset I_j^*$, i.e., there exists some element e_j such that $e_j \in I_j^*$, $e_j \notin I_j^{**}$. Since $e_j \in I_j^* = \pi_{\mathbf{I}_j}(\sigma_{C \wedge \mathbf{A}=o}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$, there exists some $(e_1, \ldots, e_{j-1}, e_{j+1}, \ldots, e_m) \in (\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_{j-1}}(I_{j-1}) \times \sigma_{C_{j+1}}(I_{j+1}) \times \ldots \times \sigma_{C_m}(I_m))$ such that $\{o\} = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(\{e_1\}) \times \ldots \times \sigma_{C_m}(\{e_m\})))$. Let us choose subsets $I_1' = \{e_1\}, \ldots, I_m' = \{e_m\}$. Since we are assuming that $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ is correct and $o \in T(I_1', \ldots, I_m')$, we would expect o to be in $T(I_1' \cap I_1^{**}, \ldots, I_m' \cap I_m^{**})$. But $T(I_1' \cap I_1^{**}, \ldots, I_m' \cap I_m^{**}) \subseteq T(\{e_1\}, \ldots, \{e_{j-1}\}, \emptyset, \{e_{j+1}\}, \ldots, \{e_m\}) = \emptyset$. Thus, $o \notin T(I_1' \cap I_1^{**}, \ldots, I_m' \cap I_m^{**})$, a contradiction. There exists no correct $\langle I_1^{**}, \ldots, I_m^{**} \rangle$ that is more precise, implying that $\langle I_1^*, \ldots, I_m^* \rangle$ is minimal. □

Theorem 6.3 follows directly from Lemmas 6.1 and 6.2. We also note that $\langle I_1^*, \ldots, I_m^* \rangle$ where each $I_i^* = \pi_{\mathbf{I}_i}(\sigma_{C \wedge \mathbf{A}=o}(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$ is proven in [16] to be the *lineage* of an SPJ transformation. Thus, we have shown via Lemma 6.2 that the lineage of an SPJ transformation given in [16] corresponds to our definition of minimal provenance (Definition 2.6). Of course our notion of minimal provenance also supports general transformations.

6.4 Augmentation

Given an SPJ transformation $T(I_1, \ldots, I_m) = \pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m)))$, recall that Theorem 6.3 only guarantees that the canonical logical specification given in Definition 6.2 encodes minimal provenance if $\mathbf{A}$ contains all attributes in C. One approach to solving this problem is to simply retain more attributes in $\mathbf{A}$ to meet the requirement, a process we call *augmentation*.

Definition 6.3 (Augmentation) Let T be a transformation with output schema $\mathbf{A}$. Transformation T' is an *augmentation* of transformation T if T' has output schema $\mathbf{A}' \supset \mathbf{A}$ and $T = T' \circ \pi_{\mathbf{A}}$. □

Augmentation can be applied to any transformation, including non-relational transformations. The benefit of augmentation is that it allows logical specifications to encode more precise provenance. However, augmentation requires more storage, and it also requires a logical or materialized view to be maintained, to produce the original output from the augmented one. Note that if augmentation retains all input attributes or even a key for each input tuple, then we effectively require each output tuple o to have a pointer to the input tuples that contributed to o. Thus, augmentation degenerates to physical provenance.

Definition 6.4 (Provenance Encoded by Augmentation) Let $O = T(I_1, \ldots, I_m)$ be a transformation instance with output schema $\mathbf{A}$. Let T' be an augmentation of T with output schema $\mathbf{A}' \supset \mathbf{A}$ and logical specification (M, F). Let $O' = T'(I_1, \ldots, I_m)$. Given $o' \in O'$, let $I^*(o') = \langle I_1^*(o'), \ldots, I_m^*(o') \rangle$ be the provenance of o' as encoded by the logical specification for T'. Given $o \in O$, the provenance of o as encoded by (M, F) is $\langle I_1^*, \ldots, I_m^* \rangle$, where each $I_i^* = \bigcup_{(o' \in O') \wedge (\pi_{\mathbf{A}}(o') = o)} I_i^*(o')$. □

Theorem 6.4 (Correctness of Provenance Encoded by Augmentation) Let $O = T(I_1, \ldots, I_m)$ be a transformation instance with output schema $\mathbf{A}$. Let T' be an augmentation of T with output schema $\mathbf{A}' \supset \mathbf{A}$ and logical specification (M, F). Given $o \in O$, the provenance of o as encoded by (M, F) is correct for o with respect to T.

Proof. Let $O' = T'(I_1, \ldots, I_m)$. Then $o \in O$ if and only if there exists $o' \in O'$ such that $\pi_{\mathbf{A}}(o') = o$. Consider any $I_1' \subseteq I_1, \ldots, I_m' \subseteq I_m$. We have:

$$
\begin{aligned}
&\{o\} \cap T(I_1', \ldots, I_m') \\
&\quad = \{o\} \cap \pi_{\mathbf{A}}(T'(I_1', \ldots, I_m')) \\
&\quad = \{o\} \cap \pi_{\mathbf{A}}(T'(I_1', \ldots, I_m') \cap \sigma_{\mathbf{A}=o.\mathbf{A}}(O')) \\
&\quad = \{o\} \cap \pi_{\mathbf{A}}(T'(I_1' \cap I_1^*, \ldots, I_m' \cap I_m^*) \cap \sigma_{\mathbf{A}=o.\mathbf{A}}(O'))
\end{aligned}
$$

$= \{o\} \cap \pi_{\mathbf{A}}(T'(I'_1 \cap I^*_1, \ldots, I'_m \cap I^*_m))$
$= \{o\} \cap T(I'_1 \cap I^*_1, \ldots, I'_m \cap I^*_m)$ □

6.5 Logical Provenance for SPJA Transformations

We now extend our results to *Select-Project-Join-Aggregate* (*SPJA*) transformations. We first introduce the *aggregation* operator.

Definition 6.5 (Aggregation) Let I be an input table I with schema $\mathbf{I}$, and let $\mathbf{G} \subseteq \mathbf{I}$, $\mathbf{B} \subseteq \mathbf{I}$. Let $g_1, \ldots, g_n$ be all of the distinct values of $\mathbf{G}$ in I, and let each group $G_i = \sigma_{\mathbf{G}=g_i}(I)$. The *aggregation* operator $\alpha_{\mathbf{G},aggr(\mathbf{B})\to V}$ takes an input table I and outputs for each group $G_i \subseteq I$ the tuple $\langle g_i, v_i \rangle$, where $v_i = aggr(G_i.\mathbf{B})$ and $aggr$ is an aggregation function on $\mathbf{B}$: $\alpha_{\mathbf{G},aggr(\mathbf{B})\to V}(I) = \bigcup_{1\leq i\leq n}\{\langle g_i, v_i\rangle\}$. We could extend to multiple aggregation functions, but the added complexity wouldn't introduce any interesting challenges. □

Definition 6.6 (SPJA Transformation) A *Select-Project-Join-Aggregate* (*SPJA*) *transformation* is a transformation that can be expressed in the canonical form $T(I_1, \ldots, I_m) = \pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m))))))$. □

In the above canonical form, $\mathbf{A}_0$ and C_0 refer to the schema $\mathbf{G} \cup \{V\}$. Note that there exist transformations that can be expressed as a tree of selection (σ), projection (π), cross-product ($\times$), and aggregation (α) operators that cannot be transformed into the above canonical form. However, any such transformation can be expressed as a tree of SPJA transformations, each of which can be transformed into the canonical form; see [30] for details. Our work considers only SPJA transformations with this canonical form, assuming more complex queries will be split into multiple transformations.

Two of the theorems for SPJ transformations have a corresponding theorem for SPJA transformations.

Theorem 6.5 Let T be an SPJA transformation: $O = T(I_1, \ldots, I_m) = \pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m))))))$. For each attribute $I_i.A \in (\mathbf{A}_0 \cap \mathbf{G})$, attribute mapping $I_i.A \leftrightarrow O.A$ holds for T.

Proof. Let $I'_1, \ldots, I'_m$ be any input set instances and let x be any possible value of A. Since $A \in \mathbf{A}_0$ and $A \in \mathbf{G}$, we have:
$\sigma_{A=x}(T(I'_1, \ldots, I'_m))$
$= \sigma_{A=x}(\pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_m}(I'_m)))))))$
$= \sigma_{A=x}(\pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\sigma_{A=x}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_m}(I'_m)))))))$
$= \sigma_{A=x}(\pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{A=x}(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_m}(I'_m)))))))$
$= \sigma_{A=x}(\pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_i}(\sigma_{A=x}(I'_i)) \times \ldots \times \sigma_{C_m}(I'_m)))))))$
$= \sigma_{A=x}(T(I'_1, \ldots, \sigma_{A=x}(I'_i), \ldots, I'_m))$ □

Theorem 6.6 Let T be an SPJA transformation: $O = T(I_1, \ldots, I_m) = \pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m))))))$. Then T has filter condition C_i on I_i.

Proof. Let $I'_1, \ldots, I'_m$ be any input set instances. Then we have:
$T(I'_1, \ldots, I'_m)$
$= \pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_i}(I'_i) \times \ldots \times \sigma_{C_m}(I'_m))))))$
$= \pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I'_1) \times \ldots \times \sigma_{C_i}(\sigma_{C_i}(I'_i)) \times \ldots \times \sigma_{C_m}(I'_m))))))$
$= T(I'_1, \ldots, \sigma_{C_i}(I'_i), \ldots, I'_m)$ □

As we did for SPJ transformations, we can generate canonical logical provenance specifications for SPJA transformations.

Definition 6.7 (Canonical Logical Specification for SPJA Transformation) Let T be an SPJA transformation: $O = T(I_1, \ldots, I_m) = \pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m))))))$. The *canonical logical provenance specification* for T is (M, F) where M contains all mappings $I_i.A \leftrightarrow O.A$ such that $I_i.A \in (\mathbf{A}_0 \cap \mathbf{G})$, and F contains all C_i. □

Based on Theorems 4.1, 6.5, and 6.6, we know that the canonical specification for SPJA transformations encodes correct provenance. For SPJA transformations T in which the outermost projection operator $\pi_{\mathbf{A}_0}$ projects out attributes in $\mathbf{G}$, we can augment T such that all attributes in $\mathbf{G}$ are retained.

Definition 6.8 (Augmentation of SPJA Transformation) Consider an SPJA transformation $T(I_1, \ldots, I_m) = \pi_{\mathbf{A}_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m))))))$ where $(\mathbf{A}_0 \cap \mathbf{G}) \subset (\mathbf{A} \cap \mathbf{G})$. The *canonical augmentation* of T is T' where $T'(I_1, \ldots, I_m) = \pi_{\mathbf{A}'_0}(\sigma_{C_0}(\alpha_{\mathbf{G},aggr(B)\to V}(\pi_{\mathbf{A}}(\sigma_C(\sigma_{C_1}(I_1) \times \ldots \times \sigma_{C_m}(I_m))))))$ and $\mathbf{A}'_0 = \mathbf{A}_0 \cup \mathbf{G}$. □

Given an augmentation T', its logical specification can contain additional attribute mappings for all attributes in $\mathbf{G}$ that are retained by $\pi_{\mathbf{A}'_0}$ but not by $\pi_{\mathbf{A}_0}$. However, even for SPJA transformations that retain all attributes in $\mathbf{G}$, the following example demonstrates that the logical specification does not always encode minimal provenance.

Example 6.1 Consider transformation **CountryAgg** with input data set SalesInfo containing country-sales pairs for each of a corporation's worldwide stores: $\mathsf{SalesInfo}(\mathsf{country}, \mathsf{sales}) = \{(France, 10), (Germany, 20), (Germany, 0)\}$. Transformation **CountryAgg** sums up the sales for each country, producing output table $\mathsf{CountrySales} = \alpha_{country,SUM(sales)}(\mathsf{SalesInfo}) = \{(France, 10), (Germany, 20)\}$. The canonical logical provenance specification is (M, F), where $M = \{\mathsf{SalesInfo.country} \leftrightarrow \mathsf{CountrySales.country}\}$ and $F = \emptyset$. Let $o = (Germany, 20) \in \mathsf{CountrySales}$. Then o's provenance as encoded by (M, F) is $\sigma_{\mathsf{country}=\text{`Germany'}}(\mathsf{SalesInfo}) = \{(Germany, 20), (Germany, 0)\}$. However, the minimal provenance of o is simply $\{(Germany, 20)\}$, since input tuple $(Germany, 0)$ has no impact on the presence or absence of o. □

In the above example, we see that input values of 0 are not in the minimal provenance of a nonzero sum in the output. Similarly, for MIN and MAX aggregates, the minimal provenance of an output tuple only contains those input tuples with the corresponding minimum or maximum value. Since the focus of our work is on logical provenance for general transformations, we do not treat individual aggregate functions here as special cases; instead, we only capture the logical provenance that can be found by treating the individual aggregate functions as black boxes.

7 System

We have built a prototype system called *Panda* (for *Provenance And Data*) that supports debugging and drill-down in workflows using either physical or logical provenance. We have previously described Panda in a demonstration description [21]. Here we focus on how Panda generates logical provenance specifications and executes our tracing algorithms.

Panda permits arbitrary data-oriented workflows with each transformation specified in either SQL or in Python. Currently, all data sets handled by Panda are encoded in relational tables, and all records (tuples) are given a globally unique ID. The main backend is **SQLite**, which stores all data sets, SQL transformations, provenance, and workflow information. For each transformation in a workflow, Panda generates logical provenance specifications consisting of attribute mappings and filters, as explained next.

7.1 SQL Transformations

Panda supports transformations specified as SQL queries, currently limited to single SELECT blocks with optional grouping and aggregation, with conjunctive conditions. Note that this form of query corresponds exactly to the SPJA transformations in Definition 6.6 of Section 6.5.

As an example, consider transformation **JoinAgg** from our running example (Figure 1). This transformation can be expressed using the following SQL query:

```
Create Table ItemCountryProfit As
Select CS.item-id, country, brand, type,
  SUM(quantity*profit_per_item) as profit,
From CustSales CS, ItemProfit IP
Where CS.item-id = IP.item-id
Group By CS.item-id, country, brand, type
```

In relational algebra, $\textbf{JoinAgg}(\mathsf{CS}, \mathsf{IP})$ can be expressed as:

$$\alpha_{item\text{-}id,country,brand,type,SUM(quantity*profit\_per\_item)\rightarrow profit}(\sigma_{CS.item\text{-}id=IP.item\text{-}id}(\mathsf{CS} \times \mathsf{IP}))$$

Panda generates logical provenance specifications through syntactic analysis of SQL queries. During syntactic analysis, Panda generates attribute mappings between attributes appearing in the SELECT clause and all possible corresponding input attributes, computed by taking the transitive closure over equalities in the WHERE clause. Panda generates filters

for all conjuncts in the WHERE clause that apply to a single input table. For our example transformation, Panda generates logical specification (M, F): $M = \{\mathsf{CS.item\_id} \leftrightarrow \mathsf{IC.item\_id}, \mathsf{CS.country} \leftrightarrow \mathsf{IC.country}, \mathsf{IP.item\_id} \leftrightarrow \mathsf{IC.item\_id}, \mathsf{IP.brand} \leftrightarrow \mathsf{IC.brand}, \mathsf{IP.type} \leftrightarrow \mathsf{IC.type}\}$, $F = \{\}$.

Now suppose the above query did not contain the item_id attribute in its SELECT clause. In this case Panda augments the query automatically as in Definition 6.7 to support logical provenance: Panda adds the item_id attribute to the SELECT clause for the purpose of provenance tracing, producing augmented result AugItemCountryProfit. It then creates a SQL view for the output data set to hide the extra attribute:

```
Create View ItemCountryProfit As
Select country, brand, type, profit,
From AugItemCountryProfit
```

To illustrate how Panda augments SPJ transformations (i.e., without aggregation), consider the following query:

```
Create Table CountryBrands As
Select country, brand
From CustSales CS, ItemProfit IP
Where CS.item-id = IP.item-id
```

In relational algebra, the above query can be expressed as:
$\pi_{country,brand}(\sigma_{CS.item\text{-}id=IP.item\text{-}id}(\mathsf{CS} \times \mathsf{IP}))$
Note that this query does not satisfy the requirements of Theorem 6.3, because the attributes in the condition $CS.item\text{-}id = IP.item\text{-}id$ are not included in the final result. Thus, Panda augments the query so that the augmented transformation satisfies the requirements of Theorem 6.3:

```
Create Table AugCountryBrands As
Select country, brand, CS.item-id
From CustSales CS, ItemProfit IP
Where CS.item-id = IP.item-id
```

Since the requirements of Theorem 6.3 are now satisfied, logical provenance for the augmented transformation encodes minimal provenance. To recover the original output, Panda creates the following view:

```
Create View CountryBrands As
Select country, brand
From AugCountryBrands
```

7.2 Python Transformations

Currently, Panda only supports Python transformations that are one-one or one-many transformations—i.e., the transformation operates on one record at a time—although this restriction can be lifted with some additional work. For Python transformations, Panda currently prefers for the user to specify logical provenance, but provides a fallback mechanism if no provenance is specified.

Consider transformation **Extract** from the running example. The user may provide Panda with the logical specification (M, F) for **Extract**: $M = \{\mathsf{CustData.cust\_id} \leftrightarrow \mathsf{CS.cust\_id}, \mathsf{CustData.country} \leftrightarrow \mathsf{CS.country}\}$, $F = \{\}$. If no user-specified logical provenance is provided, Panda augments each output record with the ID of the corresponding input record (calling it src_id in the output), and generates the attribute mapping $\mathsf{I.ID} \leftrightarrow \mathsf{O.src\_id}$ for the augmented transformation. (ID is the column for the globally unique ID present in every Panda record.) As usual, Panda stores the output of the augmented transformation, and creates a view on the augmented output to produce the original output. Note that since each record in the augmented output effectively contains a pointer to its contributing input record, augmenting the output in this case degenerates to effectively storing physical provenance.

7.3 Provenance Tracing

Panda enables provenance to be traced along a specified *tracing path*: a *source data set* O, and a *target data set* I reached via a single specified path backwards from the source. To perform tracing, Panda generates and executes a SQL query that joins multiple tables, returning the same provenance as in Algorithm 5.1.

In more detail, given a record in O, Panda first combines logical provenance wherever possible along the tracing path from O to I. Then Panda generates a query that joins the data sets in the remaining combined path. The conditions in the join query are based on the logical provenance (attribute mappings and filters) for the data sets in the combined path, and on the unique ID of the tuple being traced.

Algorithm 7.1 (Provenance Tracing) Consider a workflow instance in which each transformation T has logical provenance specification (M^T, F^T). Let $o \in O$ be the output element we want to trace through $Path = \langle I_1, I_2, \ldots, I_n, I_{n+1} \rangle$ where $I_{n+1} = O$. Using helper functions *CombinePath* and *GenerateQuery*, PT returns a query that traces the provenance of o.

$PT(o \in O, Path)$:
  for i in $[1, n]$:
    let T_i be the transformation with input set I_i and output set I_{i+1}
    let M_i denote the subset of mappings $A \leftrightarrow B$ in M^{T_i} such that
      A is from I_i and B is from I_{i+1}
    let F_i denote the subset of filters in F^{T_i} that are on I_i

let $ProvPath = \langle I_1, M_1, F_1, I_2, M_2, F_2, \ldots, I_n, M_n, F_n \rangle$
let $CPath = CombinePath(ProvPath)$
return $GenerateQuery(CPath, o \in O)$

// Combine logical provenance in *ProvPath*
CombinePath(*ProvPath*) :
suppose $ProvPath = \langle I_1, M_1, F_1, I_2, M_2, F_2, \ldots, I_n, M_n, F_n \rangle$
$CPath := \langle\rangle$
$M := \emptyset$
for i in $[n, n-1, \ldots, 2, 1]$
if $M = \emptyset$ **then**:
$M := M_i$
else:
if every right attribute in M_i is a left attribute of M **then**:
suppose $M_i = \{A_1 \leftrightarrow B_1, \ldots, A_r \leftrightarrow B_r\}$
suppose $M = \{B_1 \leftrightarrow D_1, \ldots, B_s \leftrightarrow D_s\}$, $r \leq s$
$M := \{A_1 \leftrightarrow D_1, \ldots, A_r \leftrightarrow D_r\}$
else:
$CPath := \langle I_i, M, F_i \rangle + CPath$
$M := M_i$
$CPath := \langle I_1, M, F_1 \rangle + CPath$
return *CPath*

// Trace combined logical provenance in *CPath* returned by *CombinePath*
GenerateQuery(*CPath*, $o \in O$) :
suppose $CPath = \langle I_1, M_1, F_1, I_2, M_2, F_2, \ldots, I_{n'}, M_{n'}, F_{n'} \rangle$
for i in $[1, n']$:
suppose $M_i = \{A_1 \leftrightarrow B_1, \ldots, A_r \leftrightarrow B_r\}$
let $C_i = ((A_1 = B_1) \wedge \ldots \wedge (A_r = B_r) \wedge F_i)$
let $\mathbf{I}_1$ be the schema of I_1
return $\pi_{\mathbf{I}_1}(\sigma_{C_1 \wedge \ldots \wedge C_{n'} \wedge O.ID = o.ID}(I_1 \times \ldots \times I_{n'} \times O))$

As an example, consider a LaptopProfit record (ID=15 say) that we want to trace to destination CustSales (recall Figure 1 in Section 1.1). The path between LaptopProfit (LP) and CustSales (CS) contains intermediate data set ItemCountryProfit (IC): $Path = \langle \mathsf{CS}, \mathsf{IC}, \mathsf{LP} \rangle$. The attribute mappings between CS and IC are $M_1 = \{(\mathsf{CS.item\text{-}id} \leftrightarrow \mathsf{IC.item\text{-}id}), (\mathsf{CS.country} \leftrightarrow \mathsf{IC.country})\}$, and the attribute mappings between IC and LP are $M_2 = \{(\mathsf{IC.item\text{-}id} \leftrightarrow \mathsf{LP.item\text{-}id}), (\mathsf{IC.country} \leftrightarrow \mathsf{LP.country}), (\mathsf{IC.brand} \leftrightarrow \mathsf{LP.brand}), (\mathsf{IC.profit} \leftrightarrow \mathsf{LP.profit})\}$. Since all mappings in M_1 have corresponding mappings in

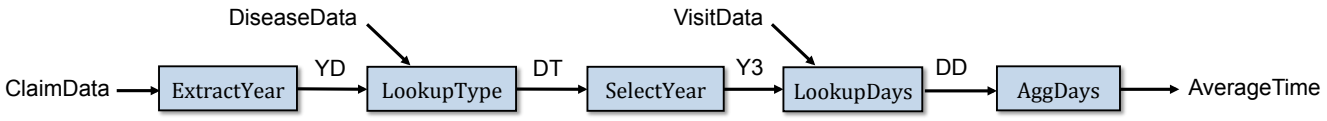

Figure 8: Health insurance data workflow.

M_2, *CombinePath* combines M_1 and M_2 to produce the following logical specification (M, F) between CS and LP: $M = \{\mathsf{CS.item\_id} \leftrightarrow \mathsf{LP.item\_id}, \mathsf{CS.country} \leftrightarrow \mathsf{LP.country}\}$, $F = \{\}$. Panda then uses *GenerateQuery* to generate the following tracing query:

```
Select Distinct CS.*
From CustSales CS, LaptopProfit LP
Where LP.item-id = CS.item-id
  And LP.country = CS.country And LP.ID = 15
```

Panda uses `Distinct` in its tracing queries so that tuples that would otherwise appear multiple times in the tracing result appear only once.

8 Experiments

The primary goal of our experiments was to explore the benefits of logical provenance over physical provenance. We will see that logical provenance has smaller time and space overhead as expected (Section 8.1), and also enables more efficient provenance tracing by combining provenance across transformations (Section 8.2). Overhead and tracing performance may vary considerably depending on the particular workflow and data. Our experiments were not designed to explore these variations (planned for future work), but rather to serve as an example of how logical provenance compares to physical provenance in one fairly neutral setting.

We conducted our experiments using the Panda system on the real-world workflow shown in Figure 8. The workflow processes health insurance claim data from the Heritage Health Prize competition [2]. The workflow's input data sets are:

- ClaimData(memberID, providerID, diseaseID, data), where attribute data is a text field containing health insurance claim data
- DiseaseData(diseaseID, type)
- VisitData(memberID, days), which contains the number of days spent in the hospital

The workflow involves the following transformations:

- **ExtractYear** extracts attribute year from ClaimData.data, producing data set YearData(memberID, providerID, diseaseID, data, year), abbreviated YD. **ExtractYear** is a Python transformation with logical specification (M, F): $M = \{(\mathsf{ClaimData.memberID} \leftrightarrow \mathsf{YD.memberID}), (\mathsf{ClaimData.providerID} \leftrightarrow \mathsf{YD.providerID}), (\mathsf{ClaimData.diseaseID} \leftrightarrow \mathsf{YD.diseaseID}), (\mathsf{ClaimData.data} \leftrightarrow \mathsf{YD.data})\}$, $F = \{\}$.

- **LookupType** joins YearData and DiseaseData on attribute diseaseID (and drops diseaseID), producing data set DiseaseType(memberID, providerID, data, year, type), abbreviated DT. **LookupType** is a SQL transformation that is augmented to keep join attribute diseaseID (recall Section 7.1). The logical specification for the augmented output DTA is (M, F): $M = \{$(YD.memberID $\leftrightarrow$ DTA.memberID), (YD.providerID $\leftrightarrow$ DTA.providerID), (YD.diseaseID $\leftrightarrow$ DTA.diseaseID),(YD.data $\leftrightarrow$ DTA.data), (YD.year $\leftrightarrow$ DTA.year), (DiseaseData.diseaseID $\leftrightarrow$ DTA.diseaseID), (DiseaseData.type $\leftrightarrow$ DTA.type)$\}$, $F = \{\}$.
- **SelectYear** applies filter year=3 to DiseaseType and drops year, producing data set Year3Data(memberID, providerID, data, type), abbreviated Y3. **SelectYear** is a SQL transformation with logical specification (M, F): $M = \{$(DT.memberID $\leftrightarrow$ Y3.memberID), (DT.providerID $\leftrightarrow$ Y3.providerID), (DT.data $\leftrightarrow$ Y3.data), (DT.type $\leftrightarrow$ Y3.type)$\}$, $F = \{$year=3$\}$.
- **LookupDays** joins Year3Data and VisitData on attribute memberID, producing DaysData(memberID, providerID, data, type, days), abbreviated DD. **LookupDays** is a SQL transformation with logical specification (M, F): $M = \{$(Y3.memberID $\leftrightarrow$ DD.memberID), (Y3.providerID $\leftrightarrow$ DD.providerID), (Y3.data $\leftrightarrow$ DD.data), (Y3.type $\leftrightarrow$ DD.type), (VisitData.memberID $\leftrightarrow$ DD.memberID), (VisitData.days $\leftrightarrow$ DD.days)$\}$, $F = \{\}$.
- Finally, **AggDays** computes the average number of days spent in the hospital for each (type, providerID) group, producing data set AverageTime(type, providerID, avgdays). **AggDays** is a SQL transformation with logical specification (M, F): $M = \{$(DD.type $\leftrightarrow$ AverageTime.type), (DD.providerID $\leftrightarrow$ AverageTime.providerID)$\}$, $F = \{\}$.

All of our experiments were run on a MacBook Air laptop (1.8 GHz Intel Core i7, 4 GB memory, 250 GB storage, Mac OS X 10.7). To measure the performance of physical provenance, we used a modified version of Panda that stores physical provenance in separate provenance tables.

8.1 Time and Space Overhead

Figure 9 demonstrates the time overhead of logical and physical provenance, running our workflow with varying input data sizes. Comparing the running times of workflow computation with and without provenance, the time overhead is roughly proportional to the size of the input data set, ranging from 3% to 6% for logical provenance, and from 45% to 62% for physical provenance. The time overhead for logical provenance is due to the additional output data produced by augmentation.

Figure 10 demonstrates the space overhead of logical and physical provenance. The space measurement totals all intermediate and output data involved in the workflow. Logical provenance incurs a consistent approximately 4% space overhead across input data sizes, while physical provenance incurs a consistent approximately 23% space overhead. Again, augmentation is responsible for essentially all of the space overhead for logical provenance.

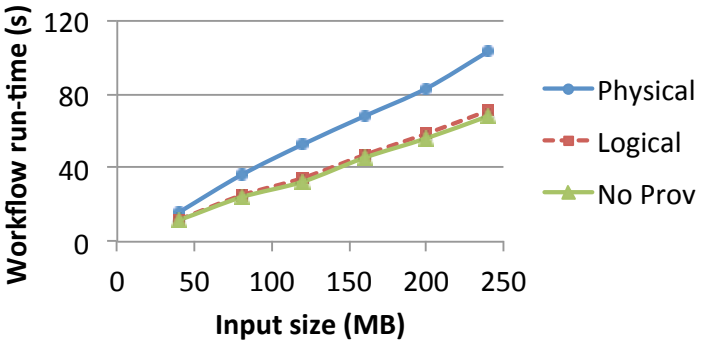

Figure 9: Time overhead of provenance capture.

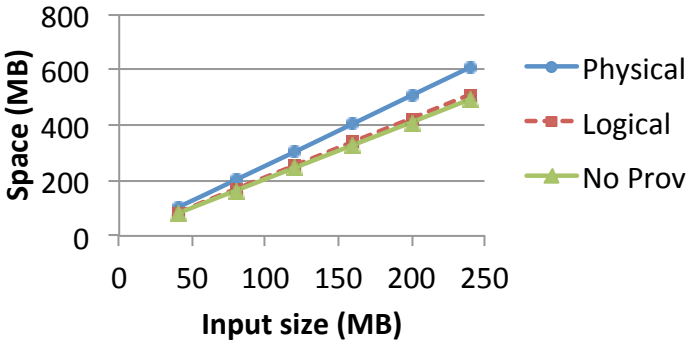

Figure 10: Space overhead of provenance capture.

8.2 Tracing Time

We measured the time to trace output tuples through our workflow to input data set ClaimData, after the workflow was run collecting physical or logical provenance. We report two times for logical provenance (Figure 11):

1. **Logical:** Time to trace one output tuple using Panda's tracing algorithm as presented in Algorithm 7.1
2. **Logical Uncombined:** Time to trace one output tuple using a tracing algorithm that does not combine provenance across transformations

For our workflow, Algorithm 7.1 combines provenance twice, eliminating intermediate data sets YearData and Year3Data from the tracing query (recall Section 5.3). Thus, while the tracing query for both physical provenance and uncombined logical provenance involves six data sets, the tracing query for combined logical provenance involves only four data sets.

As shown in Figure 11, tracing using uncombined logical provenance is 6-16% slower than using physical provenance. While both approaches involve tracing queries with the same number of data sets, tracing physical provenance is faster, since the data tables used in the tracing query for logical provenance are larger than the physical provenance tables. However, combining logical provenance in Algorithm 7.1 enables faster tracing; tracing using combined logical provenance is 32-36% faster than using physical provenance.

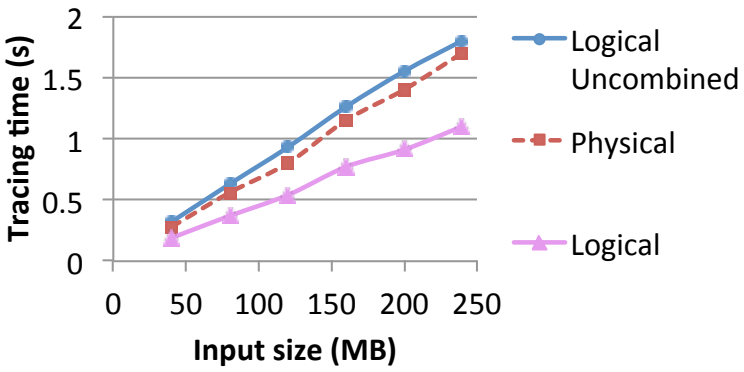

Figure 11: Provenance tracing time.

These experiments are preliminary, and our performance results only serve as one example of how logical provenance compares to physical provenance. A more comprehensive exploration of the performance tradeoffs between logical and physical provenance is left to future work (Section 9).

9 Conclusions

We presented a new general definition of provenance for transformations, introducing the notions of correctness, precision, and minimality. We determined when correctness, minimality, and weak correctness carry over from the individual transformations' provenance to the workflow provenance. We described a simple logical-provenance specification language consisting of attribute mappings and filters, and we provided tracing algorithms in workflows where logical provenance for each transformation is specified using our language. We considered logical provenance in the relational setting, showing that for a class of Select-Project-Join (SPJ) transformations, logical provenance specifications encode minimal provenance. We described our prototype system supporting the features and algorithms presented in the paper, and we reported a few preliminary experimental results.

One area of future work is to study the relationship between the precision and performance of tracing algorithms, perhaps identifying classes of workflows for which specialized tracing algorithms can greatly improve performance without losing much precision. We also plan to more comprehensively explore the performance tradeoffs between logical and physical provenance, which likely depend on the properties of the workflow and data.

Another area of future work is to explore other possible logical-provenance specification languages. Our language was designed to be expressive yet simple, but there may exist other specification languages that work especially well for workflows in particular domains.

References

[1] `http://i.stanford.edu/panda`.

[2] `http://www.heritagehealthprize.com`.

[3] The Open Provenance Model — Core Specification (v1.1). Dec. 2009. http://eprints.ecs.soton.ac.uk/18332/.

[4] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting lipstick on pig: enabling database-style workflow provenance. In *VLDB*, 2012.

[5] M. K. Anand, S. Bowers, and B. Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *EDBT*, 2010.

[6] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[7] D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *VLDB*, 2004.

[8] O. Biton, S. Cohen-Boulakia, S. B. Davidson, and C. S. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, 2008.

[9] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1), 2005.

[10] S. Bowers, T. M. McPhillips, and B. Ludäscher. Provenance in collection-oriented scientific workflows. *Concurrency and Computation: Practice and Experience*, 20:519–529, April 2008.

[11] P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[12] A. P. Chapman, H. V. Jagadish, and P. Ramanan. Efficient provenance storage. In *SIGMOD*, 2008.

[13] J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.

[14] L. Chiticariu and W.-C. Tan. Debugging schema mappings with routes. In *VLDB*, 2006.

[15] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal*, 12(1), 2003.

[16] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM TODS*, 25(2), 2000.

[17] B. Glavic and G. Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In *ICDE*, 2009.

[18] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007.

[19] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[20] T. Heinis and G. Alonso. Efficient lineage tracking for scientific workflows. In *SIGMOD*, 2008.

[21] R. Ikeda, J. Cho, C. Fang, S. Salihoglu, S. Torikai, and J. Widom. Provenance-based debugging and drill-down in data-oriented workflows. In *ICDE*, 2012.

[22] R. Ikeda, H. Park, and J. Widom. Provenance for generalized map and reduce workflows. In *CIDR*, 2011.

[23] R. Ikeda, S. Salihoglu, and J. Widom. Provenance-based refresh in data-oriented workflows. In *CIKM*, 2011.

[24] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18:1039–1065, August 2006.

[25] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. In *VLDB*, 2011.

[26] P. Missier, K. Belhajjame, J. Zhao, M. Roos, and C. Goble. Data lineage model for Taverna workflows with lightweight annotation requirements. In *IPAW*, 2008.

[27] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[28] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A not-so-foreign language for data processing. In *SIGMOD*, 2008.

[29] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3), 2005.

[30] J. D. Ullman. *Database and Knowledge-base Systems (Vol 2)*. Computer Science Press, 1989.

[31] Y. Velegrakis, R. J. Miller, and J. Mylopoulos. Representing and querying data transformations. In *ICDE*, 2005.

[32] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *ICDE*, 1997.

[33] M. Zhang, X. Zhang, X. Zhang, and S. Prabhakar. Tracing lineage beyond relational operators. In *VLDB*, 2007.