Bulletin of the Technical Committee on

Data Engineering

December 2007 Vol. 32 No. 4 **IEEE Computer Society**

Letters

Special Issue on Data Provenance

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

There are two Data Engineering Bulletin web sites: http://www.research.microsoft.com/research/db/debull and http://sites.computer.org/debull/.
The TC on Data Engineering web page is http://www.ipsi.fraunhofer.de/tcde/.

Letter from the Editor-in-Chief

Bulletin Announcement

I take great pleasure in announcing that all issues of the Data Engineering Bulletin, dating back to 1977, are now available in pdf format via the Bulletin web sites (`http://sites.computer.org/debull/` and `http://research.microsoft.com/research/db/debull/default.htm`).
Further, these issues are now also referenced via the DBLP web site as well at
`http://www.informatik.uni-trier.de/~ley/db/journals/debu/index.html`.
I believe you will find many articles that are of great interest. Some of these articles have continued to be cited many years after their publication but have been hard to acquire until now.

Many people contributed in this effort. I want first to thank the Microsoft Corporation through which the scanning of the issues was accomplished. Next, I must thank the many people who generously contributed issues of the Bulletin from their "archives" so that we can bring to you a complete set of issues. These folks are Phil Bernstein, Umesh Dayal, Stavros Christodoulakis, Mike Franklin, Hank Korth, Guy Lohman, Amihai Motro, Timos Sellis, Gio Wiederhold, and Antoni Wolski. I want to single out for special thanks Sylvia Osborn, who provided many of the very early issues that were particularly hard to find.

Finally, I want to mention that all issues of the Bulletin are now included in and accessible via DBLP. I want to thank Michael Ley for making this happen, and in an amazingly short time so that this information can be included in the announcement.

I would urge you all to visit the Bulletin web site and browse the earlier issues. I think you will be impressed by how interesting and relevant many of the papers continue to be.

The Current Issue

Database provenance is a topic that has more or less escaped attention by our field historically. We were, as a field, pre-occupied with providing the basic data storage and retrieval functionality, ensuring that performance was adequate, and generalizing to deal with issues such as additional data models, distribution, etc. One might characterize this as needing to crawl before one can walk. But our users want us now to walk. They need to know how reliable the data is, who is vouching for it, what are the uncertainties, how precise it is, whether it is under copyright, etc. To deal with this, we need to know where the data has come from and how it was derived. That is, we need to know its provenance.

Dan Suciu together with Peter Buneman have assembled the current issue of the Bulletin on the subject of "Data Provenance". The papers in this issue were carefully selected, some coming from a recent workshop on data provenance principles. I want to thank Dan and Peter for their fine job as editors of this issue, which makes data provenance much more accessible to our community. It serves as a great introduction to an important and suprisingly subtle area.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editors

As the Web makes it increasingly easy to exchange, copy and transform data, the issue of *provenance* – where data had come from and how it was derived – has rapidly become a leading research issue. Provenance has always been important in scholarship, and it is now becoming important to scientists who deal with large and complex data sets; but we do not need scholars or scientists to tell us of its importance. Go to the Web and search for the population of some country. You may well find that it is impossible to find out where a figure came from or how it was derived. derived.

The importance of provenance has been recently been recognized. The International Annotation and Provenance Workshop is devoted largely to *workflow* provenance; there is also a growing body of database research into *data* provenance The first paper in this issue by Wang-Chiew Tan, provides an accessible and comprehensive introduction to these two aspects of provenance. Data provenance has recently emerged as being important to a number of areas of computer science such as annotation, data integration, probabilistic databases, file synchronization, data archiving and program debugging. In May 2007, James Cheney, Nathan Foster and Bertram Ludäscher organized an informal workshop on the Principles of Provenance at the University of Pennsylvania in May 2007. Its purpose was to bring together people working in these areas in order to elicit some underlying principles and models. This issue is based on some of the talks at that workshop.

Provenance and Data Synchronization, by Nathan Foster and Grigoris Karvounarakis, describes two applications involving data replication: the data synchronization system Harmony where provenance tagging is needed to provide independence from order and the data sharing system Orchestra where provenance is important both for trust and for incremental updates. Following this musical demonstration, in *Program slicing and data provenance* James Cheney shows a connection between program slicing where – for debugging purposes – one wants to focus on that part of the program that influenced a specific variable and data provenance where one is interested in that part of the evolution of a database that accounts for the current state of a specific data item.

Update languages, especially the update fragment of SQL, are often dismissed by database researchers because, when measured by their ability to effect transformations of the database, they are less expressive than query languages. In *Recording Provenance for SQL Queries and Updates*, Stijn Vansummeren and James Cheney point out that when provenance is taken into account, update languages can become interesting because they can express more than query languages. This paper describes both an implicit and explicit semantics for provenance for both query and updates in SQL-like languages.

Although the focus of this issue is more on data provenance than workflow provenance, two of our papers argue forcefully that these two topics should not be divorced. The first, *Issues in Building Practical Provenance Systems* by Adriane Chapman and H.V. Jagadish, provides a set of desiderata for provenance recording and describes how simply recording workflow execution may not be adequate for understanding the evolution of data, especially when this has been heavily manipulated. The second, *Provenance in Scientific Workflow Systems* by Susan Davidson *et al* describes an an approach to summarizing workflows and then – through a collection-oriented model of data – analyzing the effect of the individual processes through a stream-based model of processing. In our opinion, connecting workflow and data provenance is the most interesting research challenge in the general field of provenance models.

Finally, in *Copyright and Provenance: Some Practical Problems*, John Ockerbloom shows how, even in the world of fixed digital documents, provenance in is important in determining intellectual property. Traditional copyright law is based on the fact that copying a work (printing it) was a non-trivial task. While traditional law may not have been ideal, it was at least workable. How we adapt it to electronic data sets in which many small pieces of data have been derived and assembled from a large number of other sources is a major challenge; and it is one in which a good model of provenance may be of significant benefit.

Peter Buneman and Dan Suciu
University Edinburgh and University of of Washington

Provenance in Databases: Past, Current, and Future

Wang-Chiew Tan*
UC Santa Cruz
wctan@cs.ucsc.edu

Abstract

The need to understand and manage provenance arises in almost every scientific application. In many cases, information about provenance constitutes the proof of correctness of results that are generated by scientific applications. It also determines the quality and amount of trust one places on the results. For these reasons, the knowledge of provenance of a scientific result is typically regarded to be as important as the result itself. In this paper, we provide an overview of research in provenance in databases and discuss some future research directions. The content of this paper is largely based on the tutorial presented at SIGMOD 2007 [11].

1 Overview of Provenance

The word *provenance* is used synonymously with the word *lineage* in the database community. It is also sometimes referred to as *source attribution* or *source tagging*. Provenance means *origin* or *source*. It also means *the history of ownership of a valued object or work of art or literature* [26]. The knowledge of provenance is especially important for works of art, as it directly determines the value of the artwork. The same applies to digital artifacts or results that are generated by scientific applications. Information about provenance constitutes the proof of correctness of scientific results and in turn, determines the quality and amount of trust one places on the results. For these reasons, the provenance of a scientific result is typically regarded to be as important as the result itself. There are two granularities of provenance considered in literature: *workflow (or coarse-grained) provenance* and *data (or fine-grained) provenance*. In what follows, we provide an overview of workflow and data provenance. However, the focus of this paper is on data provenance, which is described in the rest of this paper (Sections 2 to 4).

Workflow (or coarse-grained) provenance: In the scientific domain, a workflow is typically used to perform complex data processing tasks. A *workflow* can be thought of as a program which is an interconnection of computation steps and human-machine interaction steps. *Workflow provenance* refers to the record of the entire history of the derivation of the final output of the workflow. The amount of information recorded for workflow provenance varies. It may include a complete record of the sequence of steps taken in a workflow to arrive at some dataset. In some cases, this entails a detailed record of the versions of softwares used, as well as the models and makes of hardware equipments used in the workflow. In addition to providing a proof of correctness

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Supported in part by NSF CAREER Award IIS-0347065 and NSF grant IIS-0430994.

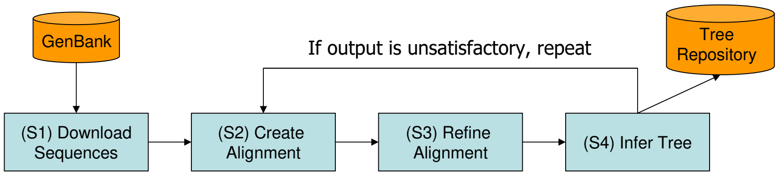


Figure 1: An example of a workflow from [16].

to the final workflow output, workflow provenance can also be useful for avoiding duplication of efforts; With appropriate bookkeeping of inputs taken by parts of the workflow, it is possible to identify parts of the workflow that need not be repeated across different execution runs.

Example 1: A simple example of a workflow from [16] is depicted in Figure 1. Arrows denote the flow of data, while boxes are used to indicate data processing steps. This workflow describes the steps taken to construct a phylogenetic tree from a set of DNA sequences. The workflow starts with step (S1) which downloads a set of DNA sequences from the GenBank repository. The second step (S2) takes the DNA sequences and runs an external sequence alignment program to generate a sequence alignment. Details of how a sequence alignment is constructed from multiple DNA sequences are "hidden" by the external program (i.e., the external program is a blackbox). Step S3 involves interaction with a biologist. The biologist examines the sequence alignment obtained from (S2) and may improve on the quality of the sequence alignment output by manually adjusting gaps inserted by the alignment program. The last step (S4) takes the edited alignment as input and produces a phylogenetic tree as output. There are in fact many steps involved in (S4) (see [16] for a detailed explanation). However, step (S4) abstracts the process of constructing a phylogenetic tree from the sequence alignment as another blackbox. The provenance of an execution of this workflow may include a record of the version of the GenBank repository used, the DNA sequences used, the software and version of the software used for sequence alignment, as well as the decisions made by the biologist in editing the alignment. □

As described in Example 1, an external process in step (S2) is involved in the workflow. In general, external processes do not possess good properties for a detailed analysis of the transformation since such details are typically hidden. Hence, the workflow provenance for this step is usually *coarse-grained*. That is, only the input, output and the software used by an external process are recorded.

Data (or fine-grained) provenance: In contrast, *data (or fine-grained) provenance* gives a relatively detailed account of the derivation of a piece of data that is in the result of a transformation step. A particular case of data provenance that is of interest to the database community and for which there have been considerable research efforts is when the transformation is specified by a database query. More precisely, suppose a transformation on a database D is specified by query Q, the provenance of a piece of data t in the output of $Q(D)$ is the answer to the following question: *Which parts of the source database D contribute to t according to Q?*

This is the subject of research of [18], where the authors described algorithms for computing data provenance in the relational framework. We give an example of data provenance in the relational setting next.

Source database D:

Employee

empid	dept
e977	CS
e132	EE
e657	BME

Department

deptid	budget
BME	1200K
CS	670K
EE	890K
MATH	230K

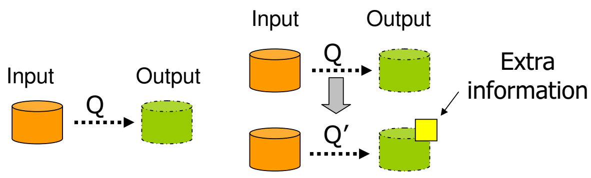
Query Q:

```
SELECT  e.empid, e.dept, d.budget
FROM    Employee e, Department d
WHERE   e.dept = d.deptid
```

Output of $Q(D)$:

empid	dept	budget
e657	BME	1200K
e977	CS	670K
e132	EE	890K

Figure 2: An example of a data transformation with SQL.



(a) (b)

	'90	...	'97	...	'00	'01	'02	'03	...	'05	'06	'07
A	[31]	... ...		... ...			[10]	[30]	... ...	[4]	[17] [21] [22]	[7] [13] [20] [23] [29]
NA		...	[32]	...	[18]	[9]			...		[14]	

(c)

Figure 3: (a) Non-Annotation (*NA*) approach, (b) Annotation (*A*) approach, (c) a timeline for data provenance research efforts using either *NA* or *A* approach.

Example 2: Suppose D and Q are the database and query, respectively, shown in Figure 2. The result of executing Q against D is also shown on the right of the same figure. The source tuples (e657, BME) and (BME,1200K) contribute to the output tuple (e657,BME,1200K) according to Q. In particular, observe that some source tuples, such as (e132,EE), play no role in contributing to the output tuple (e657,BME,1200K) according to Q. The basic idea behind the algorithms proposed in [18] is to compute data provenance by analyzing the underlying algebraic structure of relational queries. Provenance is computed and aggregated according to the underlying algebraic operators used in query on an operator-by-operator basis. □

Data provenance is the focus of this paper and we shall elaborate more on data provenance in subsequent sections. Readers who are interested in workflow provenance may find the following references useful: A survey of provenance research related to scientific data processing and scientific workflow systems [5, 19] and a survey on provenance research in E-science [28].

All of the existing research efforts on data provenance adopt one of two contrasting approaches for computing data provenance: (1) *Non-annotation approach* vs. (2) *Annotation approach*. Techniques for computing data provenance that use the non-annotation approach allow the execution of a transformation function Q as it is. See Figure 3(a), which shows a normal execution of Q. That is, Q is executed on an input database to generate an output database. In order to compute the provenance of a piece of output data, it is typically the case that the input and output database, as well as the definition of Q, are analyzed to arrive at an answer. In contrast, techniques for computing provenance that use the annotation approach (see Figure 3(b)) carry additional information to the output database. In order to compute the extra information, the original transformation Q is usually modified to another transformation Q' so that when Q' is applied on the input database, it generates an output database that is identical to that generated by Q applied on the input database, as well as the additional information. With this approach, the provenance of a piece of output data can typically be derived by analyzing the extra information.

A timeline where the research efforts are classified according to the two approaches is tabulated in Figure 3(c). We shall discuss past research efforts (i.e., mainly research efforts prior to 2005) and current research efforts (i.e., mainly research efforts between 2005 and 2007) on data provenance in Sections 2 and 3 respectively. We conclude with some future research directions in Section 4.

2 Data Provenance: Past

As described in Section 1, the problem of computing data provenance in the relational framework was studied in [18]. It is easy to see that the need to compute data provenance applies not only to tuples of relations, but also to data that occur at different levels of granularities in a hierarchical or tree-like structure. This observation was made in [9], where the authors described a hierarchical data model and an associated query language for manipulating data in this hierarchical model. The provenance of a piece of data, at any granularity, in the result of a monotone query can be obtained by analyzing the syntax of the query. In [9], the authors also made a distinction between *why* and *where-provenance*. The type of provenance studied by [18] is essentially why-provenance. Where-provenance, on the other hand, is a description of the locations in the input database which contain values where a piece of output data is copied from. To see the difference between why and where-provenance, consider Example 2 again. The why-provenance of the output tuple (e657,BME,1200K) according to Q consists of the two source tuples as described in Example 2. However, the where-provenance of the value "BME" in the output tuple (e657,BME,1200K) is the location pointed by the dept attribute in the source tuple (e657,BME). In other words, the "BME" value of the output tuple is copied from the "BME" value of the source tuple (e657,BME) and not from the "BME" value of the source tuple (BME,1200K). This is because the query Q extracts "BME" from e.dept (and not d.deptid). Observe that in contrast to the why-provenance of the output tuple (e657,BME,1200K), the where-provenance of "BME" of the same output tuple completely disregards the source Department relation.

Prior to [9] and [18], there has been a few similar research efforts [31, 32] targeted at resolving the data provenance problem. The authors of [32] proposed to build the functionality of computing data provenance into a database system using the non-annotation approach. Their motivation for using the non-annotation approach was to support provenance tracing in a database visualization environment, where large datasets are usually involved. It is therefore infeasible to associate additional information to every datum in these datasets for computing provenance. The main idea in [32] was to allow a user to register data processing functions and their corresponding inverse functions in a database system. When given a specific piece of output data to invert, an inversion planner module within the database system would infer which inverse function to apply and construct an execution plan by invoking the appropriate functions in the database system. However, since not all functions are invertible, a user is also allowed to register *weak inverses* instead. Intuitively, a weak inverse is an inverse function that approximates provenance; It may only return a subset of the desired provenance or more than what is required. A separate verification function is required to examine that the answers returned by the weak inverse are indeed answers to provenance. A fundamental drawback of this technique is that the user is required to provide (weak) inverse functions and their corresponding verification functions. Subsequent research efforts by [9] and, respectively, [18] that were described earlier, overcome this limitation by computing data provenance through analyzing the syntax and, respectively, algebraic structure of the queries.

The work of [31] first made the idea of using an annotation approach to compute provenance explicit. They proposed a polygen model ("poly" for "multiple" and "gen" for "source") that is able to track which originating data sources and intermediate data sources were used to generate an output data of interest. In [31], operational definitions on how one can compute the originating sources and intermediate sources of an attribute value over basic relational algebra operators were given.

The polygen idea was followed up by [10], where a similar set of operational definitions (called *propagation rules* in [10]) for basic relational operators were given[1]. In [10], however, the authors made clear that annotations (and not only originating sources) associated with source data can be propagated from source to output based on the propagation rules. Furthermore, the propagation rules were designed to propagate annotations based on where data is copied from (i.e., where-provenance). In particular, the relationships between *locations* of data in the input and output database were formalized through the propagation rules given in [10]. One of the problems studied in [10] is the *annotation placement problem*: *Given a query Q, source database D, a view $V = Q(D)$, and an annotation, denoted as $*$, placed in the view V, decide whether there is location to place the annotation $*$ in D so that $*$ propagates to the desired location in V and nowhere else.* If such a placement of $*$ in D exists, it is called a "side-effect-free annotation". The study of the annotation placement problem is important for understanding the bidirectional transport of annotations between the source database and the view. The authors showed a dichotomy in the complexity of the annotation placement problem for Select-Project-Join-Union (SPJU) queries: It is NP-hard to decide if there is a side-effect-free annotation for a project-join relational

[1] In [10], the authors had natural join instead of cartesian product in the set of basic relational algebra operators.

query even in the special case where the join is always performed before projection. On the other hand, there is a polynomial-time algorithm for deciding whether there is a side-effect-free annotation for SPJU queries which do not simultaneously contain both project and join operators. In fact, the annotation placement problem was later shown to be DP-hard in [30]. In [17], the authors showed that many of the complexity issues disappear for key-preserving operations, which are operations that retain the keys of the input relations.

3 Data Provenance: Current

In this section, we describe research efforts that mainly occur between 2005 and 2007, as shown in Figure 3(c). Our discussion will center around two research projects, DBNotes [4, 15] and SPIDER [1, 14], recently developed at UC Santa Cruz.

3.1 DBNotes

The work of DBNotes builds upon ideas developed in [10, 30]. DBNotes is an annotation management system for relational database systems. In DBNotes, every attribute value in a relation can be tagged with multiple annotations. When a query is executed against the database, annotations of relevant attribute values in the database are automatically propagated to attribute values in the result of the query execution. The queries supported by DBNotes for automatic annotation propagation belong to a fragment of SQL queries that corresponds roughly to select-project-join-union queries. In its *default* execution mode, annotations are propagated based on where data is copied from (i.e., where-provenance). As a consequence, if every attribute value in the database is annotated with its address, the provenance of data is propagated along, from input to output, as data is transformed by the query. An example of annotations propagated in the default manner is shown below:

Source database D:

Employee

empid	dept
e977 (a_1)	CS (a_2)
e132 (a_3)	EE (a_4)
e657 (a_5)	BME (a_6)

Department

deptid	budget
BME (b_1)	1200K (b_2)
CS (b_3)	670K (b_4)
EE (b_5)	890K (b_6)
MATH (b_7)	230K (b_8)

Query Q:

SELECT	e.empid, e.dept, d.budget
FROM	Employee e, Department d
WHERE	e.dept = d.deptid
PROPAGATE	default

Output of $Q(D)$:

empid	dept	budget
e657 (a_5)	BME (a_6)	1200K (b_2)
e977 (a_1)	CS (a_2)	670K (b_4)
e132 (a_3)	EE (a_4)	890K (b_6)

In this example, every attribute value in the source relations, Employee and Department, is annotated with a unique identifier. For instance, the attribute value 670K is annotated with the identifier b_4. The query Q has an additional "PROPAGATE default" clause, which means that we are using the default execution mode as explained earlier. By analyzing the annotations that are propagated to $Q(D)$, we can conclude that the value "BME" in $Q(D)$ was copied from "BME" in the Employee relation (and not the "BME" in Department relation). If the SELECT clause of Q had been "e.empid, d.deptid, d.budget" instead, then the annotation associated with "BME" in $Q(D)$ would be b_1 instead of a_6. Hence, equivalent queries may propagate annotations differently. This presents a serious difficulty as it means that the annotations (or provenance answers) that one obtains in the result is dependent on the query plan chosen by the database engine. DBNotes resolves this problem through a novel propagation scheme, called the *default-all* propagation scheme. In this scheme, all annotations of every equivalent formulation of a query are collected together. Consequently, propagated annotations are invariant across equivalent queries. This scheme can thus be viewed as the most general way of propagating annotations. In our example, the "BME" value in $Q(D)$ will consist of both annotations a_6 and b_1 under the default-all scheme. At first sight, the default-all scheme seems infeasible because the set of all equivalent queries is infinite in general. In [4], a practical and novel implementation that avoids the enumeration of every equivalent query is described. The key insight is that for monotone relational queries, all relevant annotations can in fact be determined by evaluating every query in a finite set of queries. Such a finite set can always be obtained, and is not too big in general. DBNotes also allows one to define *custom* propagation schemes. In this scheme, the user can specify where annotations should be retrieved from input relations. The custom scheme is especially useful when the user is, for example, only interested in retrieving annotations from a particular database, perhaps due to its authority, over other databases. In [15], the query language of DBNotes was extended to allow querying of annotations. Techniques were also developed to explain the provenance and flow of data through query transformations via the analysis of annotations.

Extensions to DBNotes. In DBNotes, as well as [10], annotations can only be placed on a column of a tuple of a relation[2]. In other words, annotations can only be associated with attribute values only, and not tuples or relations. In [21], an extension is made so that annotations can be placed on any subset of attributes of a tuple in a relation. A *color algebra* that can query both values and annotations is also described. They showed that for unions of conjunctive queries, the color algebra is complete with respect to *color relational algebra queries*. A *color relational algebra query* is a query that when applied on an color database (i.e., relations with extra columns for storing annotations) returns another color database. They also showed that every operator in the color algebra is necessary for the completeness result to hold. In [20], a similar completeness result is proven for full relational algebra instead of unions of conjunctive queries; The color algebra of [20] is shown to be complete with respect to color relational algebra queries.

In [29], the idea of associating annotations with data is further extended to allow annotations to be placed on an arbitrary collection of data in a database. A query is used to capture the collection of data of interest and the query is then associated with the desired annotations in a separate table. Similarly, one can associate a collection of data with another collection of data by using two queries that capture the collections of data of interest respectively, and then associating the queries together in a separate table.

Expressivity of languages that propagate annotations. Since many languages that manipulate annotations (or provenance) were proposed, a natural question is the comparative expressive power of these query languages. For example, one natural question is the following: How does the propagation scheme for originating sources as proposed in [31] compare with the default propagation scheme of DBNotes? Is one more expressive than the other? The work of [7] addressed this question. They showed that the default propagation scheme of DBNotes is as expressive as the propagation scheme for originating sources proposed in [31]. To show this result, they defined a query language that manipulates annotations as "first-class citizens", and showed that the propagation schemes of [31] and DBNotes are equivalent in expressive power to a certain class of queries in their language.

3.2 SPIDER

In this section, we describe a recent work on computing provenance over schema mappings that uses the non-annotation approach.

Schema mappings are logical assertions of the relationships between an instance of a source schema and an instance of the target schema. They are primary building blocks for the specification of data integration, data exchange and peer data management systems. A fundamental problem in integration systems is the design and specification of schema mappings, which typically takes a lot of time and effort to get it right [3, 24].

SPIDER [1, 14] is a system that facilitates the design of mappings by allowing mapping designers to understand, debug, and refine schema mappings at the level of mappings, through the use of examples. The idea behind SPIDER is very much like debuggers for programming languages which allow programmers to understand, debug, and refine their programs by running their programs on some test cases. The main approach that SPIDER uses to explain the semantics of mappings is through descriptions of the provenance (resp. flow) of data in the target instance (resp. source instance) through chains of possibly recursive mappings. These descriptions are called *routes* and intuitively, they describe how data in the source and target instances are related and constrained via mappings. In SPIDER, a mapping designer can either display routes ending at selected target data (i.e., trace the provenance of target data) or display routes starting at selected source data (i.e., trace the flow of source data). We describe an example of routes next.

[2]The same applies to [31], where originating sources are associated with attribute values.

Source instance I:

CardHolders

accNo	limit	ssn	name
123	\$15K	ID1	Alice

Dependents

accNo	ssn	name
123	ID2	Bob

Mappings:

for c in CardHolders ⇒ exists a in Accounts and cl in Clients where a.AccNo = c.AccNo and a.accHolder = c.ssn and cl.ssn = c.ssn and cl.name = c.name (m_1)

for d in Dependents ⇒ exists cl in Clients where cl.ssn = d.ssn cl.name = d.name (m_2)

for cl in Clients ⇒ exists a in Accounts where a.accHolder = cl.ssn (m_3)

Target instance J:

Accounts

accNo	creditLine	accHolder
123	L_1	ID1
A_1	L_2	ID2

Clients

ssn	name
ID1	Alice
ID2	Bob

The source schema consists of two relational schemas, CardHolders and Dependents, and the target schema consists of two relational schemas, Accounts and Clients. There are three mappings, m_1, m_2 and m_3, written in a query-like notation as shown in the middle of the figure above. Intuitively, the first mapping m_1 asserts that for every tuple in the CardHolders relation, there exists a tuple in the target Accounts relation and a tuple in Clients whose corresponding accNo, accHolder, ssn and name values are equal to the accNo, ssn, ssn, and name values, respectively, of the Cardholders tuple. The mapping m_2 asserts that every Dependents tuple has a corresponding Clients tuple whose ssn values coincide. The last mapping m_3 is a constraint on the target instance that says that the existence of a Clients tuple implies the existence of an Accounts tuple where the ssn value of the former is equal to the accHolder value of the latter tuple.

Given the schemas and the mappings, a mapping designer may wish to understand the mappings by executing them against a source instance I shown on the left of the figure above. A target instance J that conforms to the target schema and also satisfies the mappings is shown on the right. Such a target instance may be obtained by executing the schemas and mappings on a data exchange system such as Clio [25] or, by directly reasoning about the semantics of the mappings. In J, the values L_1, L_2 and A_1 represent possibly distinct unknown values for credit limits and account number. Since an account cannot be created without an account number, a mapping designer may probe on A_1 to understand how A_1 was formed in the exchange process. In response to the probe, SPIDER displays a route (shown below), starting from a source tuple in I that ultimately leads to the target tuple in J that contains the probed A_1.

$$\text{Dependents}(123,\text{ID2},\text{Bob}) \xrightarrow{m_2} \text{Clients}(\text{ID2},\text{Bob}) \xrightarrow{m_3} \text{Accounts}(A_2,L_2,\text{ID2})$$

Intuitively, the route explains that the Dependents tuple (i.e., Bob) in I leads to the Clients tuple (i.e., Bob) in J via mapping m_2, which in turn leads to the ID2 Accounts tuple in J via mapping m_3. Although not illustrated here, SPIDER also displays the bindings of variables in m_2 and m_3 that were used to derive each tuple in the route. By analyzing the route, a mapping designer may realize that the account number 123 in the Dependents tuple was somehow not copied over to the target and may hence refine or correct the mappings in the process.

The example above was kept simple for ease of exposition. In reality, mappings are usually not as simple as those shown in this example. They are usually larger and typically more complex. A major difficulty in computing routes is to reason about chains of possibly recursive mappings among schemas. Furthermore, the number of routes to illustrate to a mapping designer in response to a single probe may be overwhelming. In [14], an algorithm for computing routes that overcomes these difficulties has been developed. Their algorithm encodes the set of all routes, even when there may be exponentially many, in a compact polynomial-size representation. A demonstration of SPIDER is described in [1].

How-Provenance. Routes are different from why-provenance in that they not only describe which input tuples contribute to the existence of an output tuple, but also *how* the input tuples lead to the existence of the output tuple. Thus, compared to why-provenance, it is a more detailed explanation of the existence of an output tuple. In a recent paper [23], the authors described a method whereby provenance can be described using a *semiring of polynomials* in the context of datalog queries and introduced the term *how-provenance*. Semirings are similar to routes of SPIDER in that they capture the input tuples that contribute to an output tuple, as well as *how* they contribute to that output tuple. For example, let $R_1(A, B)$ be a binary relation with three tuples t_1, t_2 and t_3, where $t_1 = (1, 2)$, $t_2 = (1, 3)$ and $t_3 = (2, 3)$ and let $R_2(B, C)$ be another binary relation with three tuples t_4, t_5 and t_6, where $t_4 = (2,3)$, $t_5 = (3,3)$ and $t_6 = (3,4)$. The result of the query $\Pi_{A,C}(R_1 \bowtie R_2)$ consists of three tuples $(1,3),(1,4),(2,3),(2,4)$. The provenance polynomial for the output tuple $(1,3)$ is $t_1t_4 + t_2t_5$, which describes that the output tuple $(1,3)$ is witnessed by t_1 and t_4 or, t_2 and t_5. On the other hand, the why-provenance of $(1,3)$ according to the query is simply the set of tuples $\{t_1, t_2, t_4, t_5\}$. Algorithms for calculating provenance semirings for datalog queries were described in [23]. In [22], an application of provenance semirings is described in collaborative data sharing: Updates that are propagated along peers carry along provenance semirings. These propagated semirings are subsequently utilized to trace the derivations of an update in order to determine whether an update should be filtered based on the trust conditions specified by participants of the data sharing system.

4 Data Provenance: Future

We have described some major research efforts in data provenance in the past two decades. In this section, we describe some possible future research directions.

Most research efforts on data provenance have focused on reasoning about the behavior of provenance and keeping track of annotations or metadata through SQL queries. While SQL queries are fundamental building blocks of many database applications, knowing how to reason about the provenance and flow of data through SQL queries alone is still insufficient for a complete end-to-end tracking of the provenance and flow of data in many database applications. For example, a Web application that is powered by a database backend may only use SQL queries to retrieve data from (or store data into) the underlying database system. Data that is retrieved may still undergo various transformations (e.g., cleansing or formatting transformations) before they are displayed on a Web page. To make matters worse, many Web applications today (e.g., mashups) are based on other Web applications where information is extracted and integrated though public application programming interfaces and appropriate programming languages. In particular, the process by which information is extracted and integrated is typically not described by SQL queries. Therefore, a major unsolved challenge for data provenance research is to provide a uniform and seamless framework for reasoning about the provenance (and flow) of data through different data transformation paradigms. We list three aspects of research on data provenance next that would make progress towards resolving this challenge.

Web applications and many other systems such as data warehouses, extract-transform-load systems behave very much like workflows, where data typically undergoes a sequence of transformations in different paradigms (e.g., SQL queries, C programs or Perl scripts.). Hence, one approach towards a solution for the above mentioned unsolved challenge is to examine whether one can combine the research efforts of workflow provenance and data provenance in a uniform manner. So far, the research efforts on workflow provenance and data provenance have been somewhat independent and disconnected. For the two threads of research to converge, extensions to the formalism for workflow provenance are needed so that nodes that represent external processes in a workflow need not be treated as a blackbox. In other words, whenever possible, one should be able to drill down and analyze the provenance of data generated by external programs, which are commonly used in workflows and typically abstracted as blackboxes by current techniques for computing workflow provenance. On the other front, techniques for computing data provenance need to be extended to handle constructs of more powerful languages (e.g., aggregates, iterators, and side-effects etc.). A recent promising research effort [12, 13] uses dependency analysis techniques, similar to program slicing and program analysis techniques from the programming language community, to analyze provenance over more complex database queries that includes relational queries with grouping and aggregates. Another approach towards a uniform framework for analyzing data provenance is to abstract different data transformation paradigms using higher-level declarative formalisms such as schema mappings. However, similar to the discussion earlier, mappings will need to be enriched to model constructs of more powerful languages such as aggregates, iterators, side-effects etc.

Another research direction that would make progress towards the unsolved challenge is to develop techniques for reasoning about or approximating the provenance of data that is generated by programs through the analysis of the "blackbox behavior" of programs. In other words, even when the details of a program may not be available, one should still be able to derive the provenance of data generated by the program to a certain extent. Methods for resolving this challenge will be extremely useful in practice because in many cases, even when the details of an external program are available, they are typically too complex to be amenable to a systematic analysis.

The last research direction concerns archiving. This is a topic that bears close relationship to provenance and has not been discussed so far in this paper. Databases and schemas evolve over time. Necessarily, a complete record of provenance entails archiving all past states of the evolving database so that it becomes possible to trace the provenance of data to the correct version of the database or trace the flow of data in a version of the database that is not necessarily the most recent. Archiving is especially crucial for scientific data, where scientific breakthroughs are typically based on information obtained from a particular version of the database. Hence, all changes or all versions of the database must be fully documented for scientific results to remain verifiable. There have been some research efforts on archiving scientific datasets [6, 8]. However, two major challenges remain: (i) The first is to provide techniques for efficiently archiving versions of databases whose schema may also evolve over time. At the same time, the structure of the archive should still retain the semantics of data and relationships between entities across different versions of data as far as possible so that the archive can be meaningfully analyzed later. (ii) The second is to provide companion techniques to efficiently recover a version of the database from the archive obtained from (i), incrementally update the archive with a new version of data, as well as provide techniques to discover or analyze temporal-related properties in the archive and how entities evolve over time.

Recently, a number of applications of provenance have emerged in the context of probabilistic databases [2], schema mappings [14], and updates [22]. These applications require extensions to prior techniques for computing provenance. An interesting research direction would be to discover whether there are other applications of provenance that would require significant extensions to existing techniques or a completely new framework for computing provenance. For example, a recent workshop on provenance [27] suggests that security, information retrieval, dataflow or extract-transform-load scenarios etc. are some potential applications to investigate.

References

[1] B. Alexe, L. Chiticariu, and W.-C. Tan. SPIDER: a Schema mapPIng DEbuggeR. In *Very Large Data Bases (VLDB)*, pages 1179–1182, 2006. (Demonstration Track).

[2] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with Uncertainty and Lineage. In *Very Large Data Bases (VLDB)*, pages 953–964, 2006.

[3] P. Bernstein and S. Melnik. Model Management 2.0: Manipulating Richer Mappings. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1–12, 2007.

[4] D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. An Annotation Management System for Relational Databases. *Very Large Data Bases (VLDB) Journal*, 14(4):373–396, 2005. A preliminary version of this paper appeared in the VLDB 2004 proceedings.

[5] R. Bose and J. Frew. Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Computing Survey*, 37(1):1–28, 2005.

[6] P. Buneman, A. Chapman, and J. Cheney. Provenance Management in Curated Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 539–550, 2006.

[7] P. Buneman, J. Cheney, and S. VanSummeren. On the Expressiveness of Implicit Provenance in Query and Update Languages. In *International Conference on Database Theory (ICDT)*, pages 209–223, 2007.

[8] P. Buneman, S. Khanna, K. Tajima, and W.-C. Tan. Archiving Scientific Data. *ACM Transactions on Database Systems (TODS)*, 29(1):2–42, 2004. A preliminary version of this paper appeared in the ACM SIGMOD 2002 proceedings.

[9] P. Buneman, S. Khanna, and W.-C. Tan. Why and Where: A Characterization of Data Provenance. In *International Conference on Database Theory (ICDT)*, pages 316–330, 2001.

[10] P. Buneman, S. Khanna, and W.-C. Tan. On Propagation of Deletions and Annotations Through Views. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems (PODS)*, pages 150–158, 2002.

[11] P. Buneman and W.-C. Tan. Provenance in Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1171–1173, 2007. (Tutorial Track).

[12] J. Cheney. Program Slicing and Data Provenance. *IEEE Data Bulletin Engineering*, December 2007.

[13] J. Cheney, A. Ahmed, and U. A. Acar. Provenance as Dependency Analysis. In *Database Programming Languages (DBPL)*, pages 138–152, 2007.

[14] L. Chiticariu and W.-C. Tan. Debugging Schema Mappings with Routes. In *Very Large Data Bases (VLDB)*, pages 79–90, 2006.

[15] L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. DBNotes: A Post-It System for Relational Databases based on Provenance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 942–944, 2005. (Demonstration Track).

[16] S. Cohen, S. Cohen-Boulakia, and S. B. Davidson. Towards a Model of Provenance and User Views in Scientific Workflows. In *Workshop on Data and Integration in Life Sciences (DILS)*, pages 264–279, 2006.

[17] G. Cong, W. Fan, and F. Geerts. Annotation Propagation for Key Preserving Views. In *ACM International Conference on Information and Knowledge Management (CIKM)*, pages 632–641, 2006.

[18] Y. Cui, J. Widom, and J. L. Wiener. Tracing the Lineage of View Data in a Warehousing Environment. *ACM Transactions on Database Systems*, 25(2):179–227, 2000.

[19] S. Davidson, S. Cohen-Boulakia, A. Eyal, B. Ludascher, T. McPhilips, S. Bowers, M. K. Anand, and J. Freire. Provenance in Scientific Workflow Systems. *IEEE Data Bulletin Engineering*, December 2007.

[20] F. Geerts and J. V. den Bussche. Relational Completeness of Query Languages for Annotated Databases. In *Database Programming Languages (DBPL)*, pages 127–137, 2007.

[21] F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: Annotating and Querying Databases through Colors and Blocks. In *International Conference on Data Engineering (ICDE)*, page 82, 2006.

[22] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update Exchange with Mappings and Provenance. In *Very Large Data Bases (VLDB)*, pages 675–686, 2007.

[23] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance Semirings. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems (PODS)*, pages 675–686, 2007.

[24] L. Haas. Beauty and the Beast: The Theory and Practice of Information Integration. In *International Conference on Database Theory (ICDT)*, pages 28–43, 2007.

[25] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 805–810, 2005.

[26] Merriam-Webster OnLine. http://www.m-w.com.

[27] Workshop on Principles of Provenance (PrOPr), November 2007. http://homepages.inf.ed.ac.uk/jcheney/propr/.

[28] Y. Simmhan, B. Plale, and D. Gannon. A Survey of Data Provenance in E-Science. *SIGMOD Record*, 34:31–36, 2005.

[29] D. Srivastava and Y. Velegrakis. Intensional Associations Between Data and Metadata. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 401–412, 2007.

[30] W.-C. Tan. Containment of Relational Queries with Annotation Propagation. In *Database Programming Languages (DBPL)*, pages 37–53, 2003.

[31] Y. R. Wang and S. E. Madnick. A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective. In *Very Large Data Bases (VLDB)*, pages 519–538, 1990.

[32] A. Woodruff and M. Stonebraker. Supporting Fine-grained Data Lineage in a Database Visualization Environment. In *International Conference on Data Engineering (ICDE)*, pages 91–102, 1997.

Provenance and Data Synchronization

J. Nathan Foster
University of Pennsylvania
jnfoster@cis.upenn.edu

Grigoris Karvounarakis
University of Pennsylvania
gkarvoun@cis.upenn.edu

1 Introduction

Replication increases the availability of data in mobile and distributed systems. For example, if we copy calendar data from a web service onto a mobile device, the calendar can be accessed even when the network cannot. In peer-based data sharing systems, maintaining a copy of the shared data on a local node enables query answering when remote peers are offline, guarantees privacy, and improves performance. But along with these advantages, replication brings complications: whenever one replica is updated, the others also need to be refreshed to keep the whole system consistent. Therefore, in systems built on replication, synchronization mechanisms are critical.

In simple applications, the replicas are just that—carbon copies of each other. But often the copied data needs to be transformed in different ways on each replica. For example, web services and mobile devices represent calendars in different formats (iCal vs. Palm Datebook). Likewise, in data sharing systems for scientific data, the peers usually have heterogeneous schemas. In these more complicated systems, the replicas behave like views, and so mechanisms for updating and maintaining views are also important.

The mapping between sources and views defined by a query is not generally one-to-one. This loss of information is what makes view update and view maintenance difficult. It has often been observed that *provenance*—i.e., metadata that tracks the origins of values as they flow through a query—could be used to cope with this loss of information and help with these problems [5, 6, 4, 24], but only a few existing systems (e.g., AutoMed [12]) use provenance in this way, and only for limited classes of views.

This article presents a pair of case studies illustrating how provenance can be incorporated into systems for handling replicated data. The first describes how provenance is used in *lenses* for ordered data [2]. Lenses define updatable views, which are used to handle heterogeneous replicas in the Harmony synchronization framework [23, 13]. They track a simple, implicit form of provenance and use it to express the complex update policies needed to correctly handle ordered data. The second case study describes ORCHESTRA [17, 19], a collaborative data sharing system [22]. In ORCHESTRA, data is distributed across tables located on many different peers, and the relationship between connected peers is specified using GLAV [16] schema mappings. Every node coalesces data from remote peers and uses its own copy of the data to answer queries over the distributed dataset. Provenance is used to perform incremental maintenance of each peer as updates are applied to remote peers, and to filter "incoming" updates according to *trust conditions*.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

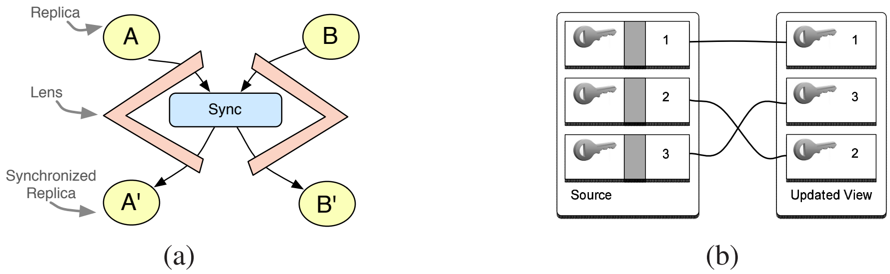


Figure 1: (a) Synchronization architecture for heterogeneous replicas. (b) Correspondence induced by keys.

2 Lenses

A *lens* is a bidirectional program. When read from left to right it denotes an ordinary function that maps sources to views. When read from right to left, the same lens denotes an "update translator" that takes a source together with an updated view and produces a new source that reflects the update.

In the context of data synchronization, lenses are used to bridge the gap between heterogeneous replicas. To synchronize two replicas represented in different formats, we first define lenses that transform each source format into a common "abstract" format, and then synchronize the abstract views. For example, to synchronize iCal and Palm Datebook calendars, we use the forward direction of two lenses to transform the files into abstract calendars, discarding the low-level formatting details and any other data specific to each replica. After synchronization, we then propagate the changes induced by the synchronizer back to the original formats using the reverse direction of the same lenses. The architecture of a synchronizer for heterogeneous data assembled in this way is depicted in Figure 1(a).

Semantically, a lens l is just a pair of functions, which we call *get* and *put*. The *get* component maps sources to views. It may, in general, discard some of the information from the source while computing the view. The *put* component therefore takes as arguments not only an updated view, but also the original source; it weaves the data from the view together with the information from the source that was discarded by the *get* component, and yields an updated source. (Note that lenses are agnostic to how the view update is expressed—the *put* function works on the entire state of the updated view.)

The two components of a lens are required to fit together in a reasonable way: the *put* function must restore all of the information discarded by *get* when the view update is a no-op, and the *put* function must propagate all of the information in the view back to the updated source (see [14] for a comparison of these requirements to classical conditions on view update translators in the literature.) In a lens language, these requirements are guaranteed by the type system; in implementations, they are checked automatically [14, 15, 3, 2].

2.1 Ordered Data

Recent work on lenses has focused on the special challenges that arise when the source and view are ordered [2]. The main issue is that since the update to the view can involve a reordering, accurately reflecting updates back to source requires locating, for each piece of the view, the corresponding piece of the source that contains the information discarded by *get*. Our solution to this problem is to enrich lenses with a simple mechanism for tracking provenance: programmers describe how to divide the source into *chunks* and generate a *key* for each chunk. These induce an association between pieces of the source and view that is used by *put* during the translation of updates—i.e., the *put* function aligns each piece of the view with a chunk that has the same key.

To illustrate the problem and our solution, let us consider a simple example from the string domain. Suppose that the source is a newline-separated list of records, each with three comma-separated fields representing the name, dates, and nationality of a classical composer, and the view contains just names and nationalities:

```
"Jean Sibelius, 1865-1957, Finnish             get      "Jean Sibelius, Finnish
 Aaron Copland, 1910-1990, American            --->      Aaron Copland, American
 Benjamin Britten, 1913-1976, English"                   Benjamin Britten, English"
```

Here is a lens that implements this transformation:

```
let ALPHA = [A-Za-z ]+
let YEARS = [0-9]{4} . "-" . [0-9]{4}
let comp = copy ALPHA . copy ", "
         . del YEARS . del ", "
         . copy ALPHA
let comps = copy "" | comp . (copy "\n" . comp)*
```

The first two lines define regular expressions describing alphabetical data and year ranges using standard POSIX notation for character sets (`[A-Za-z ]` and `[0-9]`) and repetition (`+` and `{4}`). Single composers are processed by `comp`; lists of composers are processed by `comps`. In the *get* direction, these lenses can be read as string transducers, written in regular expression style: `copy ALPHA` matches `ALPHA` in the source and copies it to the view, and `copy ", "` matches and copies a literal comma-space, while `del YEARS` matches `YEARS` in the source but adds nothing to the view. The union (`|`), concatenation (`.`), and iteration (`*`) operators work as usual. The *get* of `comps` either matches and copies an empty string or processes a each composer in a newline-separated list using `comp`. (For formal definitions see [2].)

The *put* component of `comps` restores the dates to each entry positionally: the name and nationality from the nth line in the abstract structure are combined with the years from the nth line in the concrete structure (using a default year range to handle cases where the view has more lines than the source.) For some simple updates this policy does a good job. For example, suppose that the update changes Britten's nationality, and adds a new composer to the end of the list. The *put* function combines

```
"Jean Sibelius, Finnish
 Aaron Copland, American              "Jean Sibelius, 1865-1957, Finnish
 Benjamin Britten, British     with    Aaron Copland, 1910-1990, English
 Alexandre Tansman, Polish"            Benjamin Britten, 1913-1976, English"
```

and yields an updated source

```
"Jean Sibelius, 1865-1957, Finnish
 Aaron Copland, 1910-1990, American
 Benjamin Britten, 1913-1976, British
 Alexandre Tansman, 0000-0000, Polish"
```

(The year range `0000-0000` is the default; it is generated from the regular expression `YEARS`.) On other examples, however, the behavior of this *put* function is highly unsatisfactory. For example, suppose instead that the update to the abstract string swaps the order of the second and third lines. Then the *put* function takes the following view (and the same source as above)

```
"Jean Sibelius, Finnish                        "Jean Sibelius, 1865-1957, Finnish
 Benjamin Britten, English     and yields       Benjamin Britten, 1910-1990, English
 Aaron Copland, American"                       Aaron Copland, 1913-1976, American"
```

where the year data has been taken from the entry for Copland and inserted into into Britten's, and vice versa! What we want, of course, is for the *put* to align the entries in the concrete and abstract strings by *matching* lines with identical name components, as depicted in Figure 1(b). On the same inputs, this *put* function yields

```
"Jean Sibelius, 1865-1957, Finnish
 Benjamin Britten, 1913-1976, English
 Aaron Copland, 1910-1990, American"
```

where the year ranges are correctly restored to each composer.

2.2 Provenance for Chunks

To achieve this behavior, the composers lens needs to be able to keep track of the association between lines in the source and view even when the update involves a reordering—i.e., it need to track *provenance*.

One way to do this would be using explicit *provenance tokens*. On this approach, each line of the source would be annotated with a unique identifier, and the *get* function would propagate these annotations from source to view. The disadvantage of this approach is that the view is no longer an ordinary string, but a string with annotations. This means that applications that take views as input, such as the data synchronizer described above, need to operate on annotated structures, which can be cumbersome.

Lenses use a simpler mechanism that eliminates the need to handle annotated structures. The set of lenses is enhanced with two new primitives for specifying the *chunks* of the source and a *key* for each chunk, and *put* functions are retooled to work on structures where the source is organized as a dictionary of chunks indexed by key, rather than the strings themselves. We call these *dictionary lenses*. Here is a dictionary lens that has the desired behavior for the composers example:

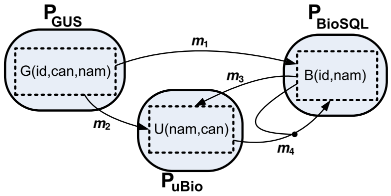
```
let comp = key ALPHA . copy ", "
         . del (YEARS . ", ")
         . copy ALPHA
let comps = "" | <comp> . ("\n" . <comp>)*
```

Compared to the previous version, the two occurrences of `comp` are marked with angle brackets, indicating that these subexpressions are the reorderable chunks, and the first `copy` at the beginning of `comp` has been replaced by the special primitive `key`. The lens `key ALPHA` copies strings just like `copy ALPHA`, but also specifies that the matched substring is to be used as the key of the chunk in which it appears—i.e., in this case, that the key of each composer's entry is their name.

The association induced by keys approximates the association that would be obtained using explicit provenance tokens. Indeed, when the keys are unique and when the view update does not modify the names, the two coincide. The idea of using keys to guide view update is not new: similar approaches have been studied in the relational setting [17]. However note that the "keys" used in dictionary lenses are not required to be keys in the strict database sense. When several pieces of the view have the same key, the *put* function pulls chunks out of the dictionary in the order that they originally appeared in the source. This gives the option of obtaining other useful update policies via the choice of key. For example, if a *put* function that operates by position is desired, it can be programmed as a lens whose key component returns a constant.

Another way to control the update policy embodied in a dictionary lens is via the definition of chunks. Many examples can be processed using one level of chunking, as in the composer lens. But chunks may also be nested, which has the effect of stratifying matching into levels: top-level chunks are matched globally across the entire string, subchunks are aligned locally within each chunk, and so on. This is useful in cases where the source has nested structure—e.g., it is used in a lens for LaTeX sources.

We have used dictionary lenses to build lenses for a variety of textual formats including vCard, CSV, and XML address books, iCal and ASCII calendars, BibTeX and RIS bibliographic databases, LaTeX documents, iTunes libraries, and protein sequence data represented in the SwissProt format and XML. These examples demonstrate that a simple notion of implicit provenance formulated using keys is capable of expressing many useful update policies. Current work is focused on an extension to key matching that uses "fuzzy" metrics such as edit distance to align chunks. This relaxed form of matching is useful when processing data with no clear key such as documents, and for handling cases where the update changes a key. We are also studying primitives that incorporate explicit metadata (e.g., source string locations) into the keys, and on developing dictionary lenses for richer structures such as trees and graphs.



$$\begin{aligned} m_1 &: \quad G(i,c,n) \rightarrow B(i,n) \\ m_2 &: \quad G(i,c,n) \rightarrow U(n,c) \\ m_3 &: \quad B(i,n) \rightarrow \exists c\, U(n,c) \\ m_4 &: \quad B(i,c) \wedge U(n,c) \rightarrow B(i,n) \end{aligned}$$

Figure 2: Example collaborative data sharing system for bioinformatics sources. For simplicity, each peer ($P_{GUS}, P_{BioSQL}, P_{uBio}$) has one relation. Schema mappings, given at the right, are indicated by labeled arcs.

3 ORCHESTRA

ORCHESTRA is a *collaborative data sharing system* (abbreviated CDSS) [22], i.e., a system for data sharing among heterogeneous peers related by a network of schema mappings. Each peer has a locally controlled and edited database instance, but wants to ask queries over related data from other peers as well. To achieve this, every peer's updates are translated and propagated along the mappings to the other peers. However, this *update exchange* is filtered by *trust conditions*, expressing what data and sources a peer judges to be authoritative, which may cause a peer to reject another's updates. In order to support such filtering, updates carry *provenance* information. ORCHESTRA targets scientific data sharing, but it can also be used for other applications with similar requirements and characteristics.

Figure 2 illustrates an example bioinformatics CDSS, based on a real application and databases of interest to affiliates of the Penn Center for Bioinformatics. GUS, the Genomics Unified Schema, contains gene expression, protein, and taxon (organism) information; BioSQL, affiliated with the BioPerl project, contains very similar concepts; and a third schema, uBio, establishes synonyms and canonical names for taxa. Instances of these databases contain taxon information that is autonomously maintained but of mutual interest to the others. Suppose that a BioSQL peer, P_{BioSQL}, wants to import data from peer P_{GUS}, as shown by the arc labeled m_1, but the converse is not true. Similarly, peer P_{uBio} wants to import data from P_{GUS}, along arc m_2. Additionally, P_{BioSQL} and P_{uBio} agree to mutually share some of their data: e.g., P_{uBio} imports taxon synonyms from P_{BioSQL} (via m_3) and P_{BioSQL} uses transitivity to infer new entries in its database, via mapping m_4. Finally, each peer may have a certain *trust policy* about what data it wishes to incorporate: e.g., P_{BioSQL} may only trust data from P_{uBio} if it was derived from P_{GUS} entries. The CDSS facilitates dataflow among these systems, using mappings and policies developed by the independent peers' administrators.

The arcs between peers are sets of *tuple-generating dependencies* (tgds). Tgds are a popular means of specifying constraints and mappings [11, 10] in data sharing, and they are equivalent to so-called *global-local-as-view* or *GLAV* mappings [16, 21]. Some examples are shown in the right part of Figure 2. For instance, m_1 says that, if there is a tuple in G about an organism with id i, canonical name c and name n, then an entry (i, n) should be inserted in B. Another mapping, m_4, ensures that, if there is an entry in B associating id i with a name c, and - according to U - n is a synonym of c, then there is also an entry (i, n) in B. Observe that m_3 has an existential variable. For such mappings, update exchange, also involves inventing new "placeholder" values, called *labeled nulls*. Figure 3(a) illustrates update exchange on our running example: assuming that the peers have the local updates shown on the top, (where '$+$' signifies insertion), the update translation constructs the instances shown on the bottom (where c_1, c_2, c_3 are labeled nulls).

3.1 Using Provenance for Trust Policies

In addition to schema mappings, which specify the relationships between data elements in different instances, a CDSS supports *trust policies*. These express, for each peer P, what data from update translation should be trusted and hence accepted. Some possible trust conditions in our CDSS example are:

- Peer P_{BioSQL} distrusts any tuple $B(i, n)$ if the data came from P_{GUS}, and trusts any tuple from P_{uBio}.

ΔG

+	1	2	3
+	3	5	2

ΔB

+	3	5

ΔU

+	2	5

G

1	2	3
3	5	2

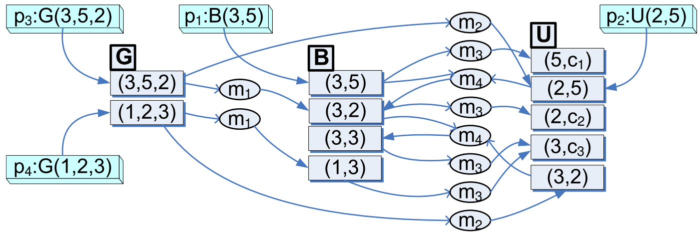
B

3	5
3	2
1	3
3	3

U

2	5
3	2
5	c_1
2	c_2
3	c_3

(a)



(b)

Figure 3: Example of update exchange and resulting provenance graph

- Peer P_{BioSQL} distrusts any tuple $B(i, n)$ that came from mapping (m_4) if $n \neq 2$.

Since the trust conditions refer to other peers and to the schema mappings, the CDSS needs a precise description of how these peers and mappings have contributed to a given tuple produced by update translation, i.e., *data provenance*. Trust conditions need a more detailed provenance model than why-provenance [6] and lineage [9, 1], as explained in [17]. Informally, we need to know not just from which tuples a tuple is derived, but also *how* it is derived, including separate alternative derivations through different mappings.

Figure 3(b) illustrates the main features of our provenance model with a graphical representation of the provenance of tuples in our running example (a more formal description can be found in [17, 18]). The graph has two kinds of nodes: tuple nodes (rectangles), and mapping nodes (ellipses). Arcs connect tuple nodes to mappings that apply to them, and mapping nodes to tuples they produce. In addition, we have nodes for the insertions from the local databases. This "source" data is annotated with its own id (unique in the system) $p_1, p_2, \ldots$ etc. (called a *provenance token*), and is connected by an arc to the corresponding tuple entered in the local instance.

Note that, when the mappings form cycles, it is possible for a tuple to have infinitely many derivations, as well as for the derivations to be arbitrarily large; nonetheless, this graph is a finite representation of such provenance. From the graph we can analyze the provenance of, say, $B(3, 2)$ by tracing back paths to source data nodes — in this case through (m_4) to p_1 and p_2 and through (m_1) to p_3. This way, we can detect when the derivation of a tuple is "tainted" by a peer or by a mapping, i.e., if all its derivations involve them, or not, if there are alternative derivations from trusted tuples and mappings. For example, distrusting p_2 and m_1 leads to rejecting $B(3, 2)$ but distrusting p_1 and p_2 does not.

3.2 Using Provenance for Incremental Update Exchange

One of the major motivating factors in our choice of provenance formalisms has been the ability to *incrementally maintain* both the data instances at every peer and the provenance associated with the data. Similarly to the case of trust conditions, the provenance model of ORCHESTRA is detailed enough for incremental maintenance, while *lineage* [9, 1] and *why-provenance* [6] are not, intuitively because they don't identify alternative derivations of tuples. We represent the provenance graph *together* with the data instances, using additional relations (see [17] for details). Schema mappings are then translated to a set of datalog-like rules (the main difference from standard datalog being that $Skolem$ functions are used to invent new values for the labeled nulls). As a result, incremental maintenance of peer instances is closely related to incremental maintenance of recursive datalog views, and some techniques from that area can be used. Following [20] we convert each mapping rule (after the relational encoding of provenance) into a series of *delta rules*.

For the case of incremental insertion, the algorithm is simple and analogous to the incremental view maintenance algorithms of [20]. Incremental deletion is more complex: when a tuple is deleted, we need to decide whether other tuples that were derived from it need to be deleted; this is the case if and only if these derived tuples have no alternative derivations from base tuples. Here, ORCHESTRA's provenance model is useful in order to identify tuples that have no derivations and need to be deleted. A small complication comes from the fact that there may be "loops" in the provenance graph, such that several tuples are mutually derivable from one another, yet none are derivable from base tuples. In order to "garbage collect" these no-longer-derivable tuples, we can also use provenance, to test whether they are derivable from trusted base data; those tuples that are not must be recursively deleted following the same procedure.

Revisiting the provenance graph of Figure 3(b), suppose that we wish to propagate the deletion of the tuple $B(3,5)$. This leads to the invalidation of mapping nodes labeled m_3 and m_4. Then, for the tuples that have incoming edges from the deleted mapping nodes, $U(5,c_1)$ has to be deleted, because there is no other incoming edge, while for $B(3,2)$ there is an alternative derivation, from $G(3,5,2)$ through (m_1), and thus it is not deleted. We note that a prior approach to incremental view maintenance, the DRed algorithm [20], has a similar "flavor" but takes a more pessimistic approach. Upon the deletion of a set of tuples, DRed will pessimistically remove all tuples that can be transitively derived from the initially deleted tuples. Then it will attempt to re-derive the tuples it had deleted. Intuitively, we should be able to be more efficient than DRed on average, because we can exploit the provenance trace to test derivability in a goal-directed way. Moreover, DRed's re-derivation should typically be more expensive than our test for derivability, because insertion is more expensive than querying, since the latter can use *only* the keys of tuples, whereas the former needs to use the complete tuples; when these tuples are large, this can have a significant impact on performance. Experimental results in [17] validate this hypothesis.

In the future, we plan to add support for bidirectional propagation of updates over mappings. In this case, we have to deal with a variation of the view update problem, and we expect provenance information to be useful in order to identify possible update policies for the sources and *dynamically* check if they have side-effects on the target of the mappings.

4 Discussion

These case studies describe some first steps towards applying provenance to problems related to data replication. In particular, they demonstrate how tracking provenance, either implicitly as in lenses or explicitly as in ORCHESTRA, can improve solutions to traditionally challenging problems such as view update and view maintenance.

There is burgeoning interest in provenance, and more sophisticated models are being actively developed. Whereas early notions such as *lineage* [9] and *why-provenance* [6] only identified which source values "contribute to" the appearance of a value in the result of a query, more recent models [7, 18] also describe *how* those source values contributes to the value in the result. We believe that as these richer models are developed, they will increasingly be applied at all levels of systems including in mechanisms for creating, maintaining, and updating views, for debugging schema mappings [7], and for curating and synchronizing replicated data.

Acknowledgements The systems described in this article were developed in collaboration with members of the Harmony (Aaron Bohannon, Benjamin Pierce, Alexandre Pilkiewicz, Alan Schmitt) and ORCHESTRA (Olivier Biton, Todd Green, Zachary Ives, Val Tannen, Nicholas Taylor) projects; the case studies are based on papers co-authored with them. We wish to thank Peter Buneman for helpful comments on an early draft. Our work has been supported by NSF grant IIS-0534592 (Foster), and NSF grants IIS-0477972, 0513778, and 0415810, and DARPA grant HR0011-06-1-0016 (Karvounarakis).

References

[1] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB 2006, Proceedings of 31st International Conference on Very Large Data Bases, September 12-15, 2006, Seoul, Korea*, pages 953–964, 2006.

[2] A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, and A. Schmitt. Boomerang: Resourceful lenses for string data. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), San Francisco, California*, Jan. 2008. To appear.

[3] A. Bohannon, J. A. Vaughan, and B. C. Pierce. Relational lenses: A language for updateable views. In *ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems, Chicago, Illinois*, 2006. Extended version available as University of Pennsylvania technical report MS-CIS-05-27.

[4] P. Buneman, A. Chapman, J. Cheney, and S. Vansummeren. A provenance model for manually curated data. In *International Provenance and Annotation Workshop (IPAW), Chicago, IL*, volume 4145 of *Lecture Notes in Computer Science*, pages 162–170. Springer, 2006.

[5] P. Buneman, S. Khanna, and W. C. Tan. Data provenance: Some basic issues. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS), New Delhi, India*, volume 1974 of *Lecture Notes in Computer Science*, pages 87–93. Springer, 2000.

[6] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In J. V. den Bussche and V. Vianu, editors, *Database Theory — ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, pages 316–330. Springer, 2001.

[7] L. Chiticariu and W.-C. Tan. Debugging schema mappings with routes. In *VLDB 2006, Proceedings of 31st International Conference on Very Large Data Bases, September 12-15, 2006, Seoul, Korea*. ACM Press, 2006.

[8] G. Cong, W. Fan, and F. Geerts. Annotation propagation revisited for key preserving views. In *ACM International Conference on Information and Knowledge Management (CIKM), Arlington, VA*, pages 632–641, 2006.

[9] Y. Cui. *Lineage Tracing in Data Warehouses*. PhD thesis, Stanford University, 2001.

[10] A. Deutsch, L.Popa, and V. Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.

[11] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336:89–124, 2005.

[12] H. Fan and A. Poulovassilis. Using schema transformation pathways for data lineage tracing. In *BNCOD*, volume 1, pages 133–144, 2005.

[13] J. N. Foster, M. B. Greenwald, C. Kirkegaard, B. C. Pierce, and A. Schmitt. Exploiting schemas in data synchronization. *Journal of Computer and System Sciences*, 73(4):669–689, June 2007. Extended abstract in *Database Programming Languages (DBPL)* 2005.

[14] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. *ACM Transactions on Programming Languages and Systems*, 29(3):17, May 2007. Extended abstract in *Principles of Programming Languages* (POPL), 2005.

[15] J. N. Foster, B. C. Pierce, and A. Schmitt. A logic your typechecker can count on: Unordered tree types in practice. In *Workshop on Programming Language Technologies for XML (PLAN-X), Nice, France, informal proceedings*, Jan. 2007.

[16] M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational plans for data integration. In *Proceedings of the AAAI Sixteenth National Conference on Artificial Intelligence, Orlando, FL USA*, pages 67–73, 1999.

[17] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB 2007, Proceedings of 32nd International Conference on Very Large Data Bases, September 25-27, 2007, Vienna, Austria*, 2007.

[18] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proceedings of the Twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, 2007.

[19] T. J. Green, N. Taylor, G. Karvounarakis, O. Biton, Z. Ives, and V. Tannen. ORCHESTRA: Facilitating collaborative data sharing. In *SIGMOD 2007, Proceedings of the ACM International Conference on Management of Data, June 11-14, 2007, Beijing, China*, 2007. Demonstration description.

[20] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, pages 157–166. ACM Press, 1993.

[21] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 505–516. IEEE Computer Society, March 2003.

[22] Z. Ives, N. Khandelwal, A. Kapur, and M. Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *CIDR 2005: Second Biennial Conference on Innovative Data Systems Research, Asilomar, CA*, pages 107–118, January 2005.

[23] B. C. Pierce et al. Harmony: A synchronization framework for heterogeneous tree-structured data, 2006. `http://www.seas.upenn.edu/~harmony/`.

[24] L. Wang, E. A. Rundensteiner, and M. Mani. U-filter: A lightweight XML view update checker. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Atlanta, GA*, page 126. IEEE Computer Society, 2006.

Program Slicing and Data Provenance

James Cheney
University of Edinburgh

Abstract

Provenance is information that aids understanding and troubleshooting database queries by explaining the results in terms of the input. Slicing is a program analysis technique for debugging and understanding programs that has been studied since the early 1980s, in which program results are explained in terms of parts of the program that contributed to the results. This paper will briefly review ideas and techniques from program slicing and show how they might be useful for improving our understanding of provenance in databases.

1 Introduction

The result of a query could be considered "incorrect" in a number of ways: the input data might be erroneous; the query might contain incorrect data values; or the query itself might be misleading or subject to misinterpretation. For example, consider the queries Q_1, Q_2, Q_3:

Q_1	`SELECT Name, Height FROM People WHERE Name = 'James'`
Q_2	`SELECT Name, '200' AS Height  FROM People WHERE Name = 'James'`

Q_3	`SELECT P.Name, Q.Weight AS Height`
	`FROM People P, People Q`
	`WHERE P.Name = 'James' AND Q.Name = 'Bob'`

Suppose that each of these queries returns the same record (Name:James, Height:200) when run against some database DB, having a table with schema People(Name, Height, Weight). We might interpret this result as saying that the person James has height 200cm; this happens to be incorrect if 'James' refers to the author of this article. However, in the first case, the error is in the *original data*; in the second case, the error is in the *query*; and in the third case, the error is the mismatch between the user's *interpretation* of the query result and what the query actually says. Of course, there are many other possible sources of error or misinterpretation, such as units of measure (e.g. centimeters versus inches) which we will not consider here.

An expert user who is familiar with the semantics of the query language and who has access to the database can, with some effort, trace erroneous query results to the underlying data in the input, and perhaps "clean" or repair the errors. A lot of recent research has been undertaken to automate the expensive process of correcting errors (or reconciling inconsistencies) in databases, often called *data cleaning* [8]. Automatic data cleaning works best when there is a clear, formal definition of "correct" or "consistent" data; in practice, correctness is

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

often taken to be consistency with keys, functional dependencies, or other database constraints. However, it is usually left to the user to determine which constraints characterize "clean" data.

The other problems of misformulation or misinterpretation of a query are more difficult to detect and correct. This problem is compounded by barriers between end-users and databases in typical systems. For example, in a typical Web application, queries are generated by middleware based on user input from a form, so the user who must interpret the results of the query is often not the author of the query, and may not have direct access to either the query or database. Thus, from such a user's point of view, the database (and the overall system) is a "black box" that accepts form input and produces results, which are presented as bare assertions without any supporting explanation or *evidence* that could be used to decide whether the results are trustworthy or not or whether the query accurately reflects the user's interpretation of the results.

There are, of course, many possible ways to bridge this gap. Previous work on *provenance* in databases (see, for example, [9] for an overview) has sought to provide such explanations, for example to answer questions about a query result such as "Why was this record part of the result?" or "Where in the input database did this value come from?" In this article, we consider provenance to be any information that explains how the results were obtained from the underlying database. However, this informal definition begs the questions: just what *is* an explanation, and what makes one explanation preferable to another?

A number of answers have been proposed in previous work on provenance. For example, in approaches such as Cui, Widom and Wiener's *lineage* [6] and Buneman, Khanna and Tan's *why-provenance*, an explanation (called "witness" in [4]) for the presence of a record t in the output of a query Q run on database DB is a subset DB' of the records in DB such that $t \in Q(DB')$. Moreover, there is a "best" explanation DB' is obtained by combining all of the *minimal* explanations. A related approach called *where-provenance* [4] records the source locations in the input from which output data were copied. Most of these definitions are sensitive to the syntax of the query, thus the provenance may be altered by query rewriting. Minimal why-provenance is insensitive to query rewriting, but it appears difficult to extend beyond monotone (`SELECT-FROM-WHERE-UNION`) queries. In particular, features such as negation, grouping, and aggregation are problematic for these techniques.

However, databases are certainly not the only setting in which it is important to be able to explain the behavior of a large system. This is a central issue in software engineering, security, and many other areas. Therefore it may be worthwhile to consider whether ideas or techniques in these other branches of computer science can be transferred to the database and data provenance settings.

Program slicing is a well-explored technique in software engineering. Intuitively, program slicing attempts to provide a concise explanation of a bug or anomalous program behavior, in the form of a fragment of the program that shows only those parts "relevant" to the bug or anomaly. There seems to be a compelling analogy between program slicing and data provenance, since most approaches to the latter propose to explain part of the result of a query using a "relevant" part of the input database. In this article, we explore this analogy and discuss a form of provenance based on ideas from program slicing and related concepts such as *dependency analysis*.

In the rest of this article, we provide some background discussion of dependency analysis and program slicing (Section 2), show how similar ideas can be used to develop a fine-grained notion of dependency provenance (Section 3), and conclude by discussing some research questions for future work (Section 4). We focus on high-level exposition rather than technical details which can be found in a recent paper [5].

2 Program slicing background

Consider the straight-line program fragment shown in Figure 1(a). If we execute this program in a context where initially $x = 1, y = 2, z = 3, w = 4$, then the final w-value of the program will be $w = 23$. If we were expecting a different result value for w, such as 17, then we might like to know what parts of the program are responsible. To diagnose the problem, it would be helpful to highlight a subset of the statements which were relevant to the final result of w, and ignore the other statements. Informally, a *slice* is a subset of the statements of the program

```
x = y + 2*z;
y = z + 3*w;
z = w - 4*x;
w = x + y;
y = z + 3*w;
z = w - 4*x;
```

(a) Program

```
x = y + 2*z;
y = z + 3*w;
...
w = x + y;
...
...
```

(b) w-slice

```
x = y + 2*z;
...
...
...
...
...
```

(c) x-slice

```
x = y + 2*z;
y = z + 3*w;
z = w - 4*x;
w = x + y;
y = z + 3*w;
...
```

(d) y-slice

```
x = y + 2*z;
y = z + 3*w;
...
w = x + y;
...
z = w - 4*x;
```

(e) z-slice

Figure 1: Straight-line program and slices with respect to w, x, y, and z

```
if (x == 0) {
  y = z + w;
  x = 10;
  w = y + 1;
} else {
  y = x + w;
  x = x - 1;
  w = 5;
}
```

(a) Program

```
if (x == 0) {
  y = z + w;
  ...
  w = y + 1;
} else {
  ...
  ...
  w = 5;
}
```

(b) Static w-slice

```
if (x == 0) {          if (x == 0) {
  ...                    y = z + w;
  ...                    ...
  ...                    w = y + 1;
} else {               } else {
  ...                    ...
  ...                    ...
  w = 5;                 ...
}                      }
```

(c) Dynamic w-slices for $x = 0$, $x \neq 0$

Figure 2: Conditional program with static and dynamic slices with respect to w

that are relevant to some part of the output. Figure 1(b) shows a slice of the program with respect to w; we have replaced the statements that do not "contribute" to the final value of w with ellipses. Similarly, Figure 1(c)–(e) depict slices with respect to x, y, and z.

Conditional expressions make the slicing problem slightly more interesting. For example, consider Figure 2(a). Since conditionals introduce the possibility of having code in the program that is not executed during a particular run, we distinguish between static and dynamic slices; the former cannot take into account the values actually encountered at run time. A *static slice* for this program with respect to w includes statements in both branches because we do not know which branch will be taken; see Figure 2(b). In a dynamic slice, we may omit all of the code in the branch that is not taken; for example, depending on whether the initial value of x is zero or nonzero, the dynamic slice for w would be as shown in the left or right of Figure 2(c), respectively.

It is, of course, trivial to find at least one program slice: the program itself. However, the goal of slicing is to aid understanding a large and complex program by identifying a small, and hopefully easy-to-understand, subset of program points. As with most interesting program properties, computing minimal slices (whether static or dynamic) is undecidable; it is intractable even if we restrict to programs with conditionals and assignment but without while-loops or recursion. Thus, in practice, program slicing techniques attempt to conservatively approximate the minimal slice.

Slicing captures an intuitive debugging process used by experienced programmers [12]. Since its introduction by Weiser [11], both static and dynamic program slicing have been investigated extensively [10]. Subsequent research has identified *dependence* as a key concept in slicing and a number of related program analysis techniques [1]. In program analysis, dependence information describes how parts of a program, such as variables or control flow points, affect other parts. This information is valuable because it can be used to predict how the program will behave statically before execution or to understand how the program actually behaved after execution. Dependences are often classified into *data dependences*, or dependences on data from which an object was computed, and *control dependences*, or dependences on data that affected the flow of control leading to the computation of an object.

While the majority of research on slicing has considered imperative (C) or object-oriented paradigms, slicing techniques have also been adapted to declarative (functional or logic) programming paradigms which are closely related to database query languages such as SQL.

3 A slicing approach to provenance

In databases, it is usually the *data* that is large and poorly understood, while the query is relatively small. Previous work on data provenance has often defined provenance as a set of "parts" of the input (e.g. fields or records) that "explains" a part of the output. There is a compelling analogy between program slicing, which uses part of a program as a concise "explanation" for part of the output, and data provenance, which uses part of the database to explain part of the output. This analogy suggests that we may be able to transfer ideas and techniques for program slicing into the database and data provenance setting. We explore this idea in the rest of the article.

Recall the queries Q_1, Q_2, Q_3 from the introduction. Suppose we run each of them on the input database consisting of the table People shown in Figure 3. This database contains just three entries. When run against this table, queries Q_1–Q_3 produce produces exactly one record, namely (James, 200).

We now might like to know: What parts of the input does the Height field in this record depend on? There are many possible answers, depending on how we interpret the term "depend". One natural notion is to consider the how a change to each part of the input affects the output. We say that a part of the output *depends on* a part of the input if changing the input part *may* result in a change to the output part. Thus, as in program slicing, we need to consider not just what actually did happen but also what might have happened: how would the output change if the input were slightly different?

We consider three kinds of dependences: dependences of output *relations*, *records*, or *fields* on field values in the input. Consider a query Q and input database I and record $s \in I$ with field B. We say that the output relation *depends on* $s.B$ if changing the value of $s.B$ may cause the output to change in any way. We say that a record $r \in Q(DB)$ *depends on* $s.B$ if changing the value of $s.B$ may delete in r from the output. Finally, we say that the field value $r.A$ in the output *depends on* $s.B$ in the input if there is some way to change the value of $s.A$ that either deletes r from the output or changes the value of $r.A$. The *dependency provenance* of $r.A$ is the set of all input fields $s.B$ on which $r.A$ depends on. Since the dependency provenance of a part of the input is a subset of fields of the input, we can think of it as being a *data slice* of the input in which irrelevant parts not in the dependency provenance are elided.

We want to emphasize that this is only an informal definition but that it can be made precise and generalized; however, here we will only illustrate the idea through examples. Recall the example from the introduction. Figures 4(a–c) show data slices of the input data u_1.Height for queries Q_1–Q_3. For Q_1, the dependency provenance of u_1.Height consists of t_1.Name and t_1.Height. The value of u_1.Height was copied from t_1.Height, and the output also depends on t_1.Name, because changing this value would make u_1 disappear from the output. For Q_2, however, as shown in Figure 4(b), u_1 does not depend on t_1.Height; the value 200 was provided by the query, not copied from the input. It does still depend on t_1.Name field for the same reason as Q_1. For Q_3, as shown in Figure 4(c), t_1.Height does not depend on t_1.Height in the input, but it *does* depend on t_3.Name and t_3.Weight.

Dependency provenance is clearly similar in some respects to previous approaches such as why-provenance, where-provenance and lineage. In particular, where-provenance (that is, the input field from which an output field was "copied") appears to be included in the dependency provenance. Moreover, for conjunctive queries like the above, the lineage (that is, the input records that "contributed" in some way to an output record) appears to

People			
id	Name	Height	Weight
t_1	James	200	190
t_2	Alice	160	150
t_3	Bob	204	200

$\Longrightarrow$

id	Name	Height
u_1	James	200

Figure 3: Input data and result of running queries Q_1, Q_2, and Q_3

id	Name	Height	Weight
t_1	James	200	$\cdots$
$\vdots$			

(a) Q_1

id	Name	Height	Weight
t_1	James	$\cdots$	$\cdots$
$\vdots$			

(b) Q_2

id	Name	Height	Weight
t_1	James	$\cdots$	$\cdots$
t_3	Bob	$\cdots$	200
$\vdots$			

(c) Q_3

Figure 4: Data slices with respect to u_1.Height and queries Q_1, Q_2, and Q_3

include all of the records mentioned in the dependency provenance. Finally, why-provenance seems very closely related, but a direct comparison is difficult because the original paper [4] used a semi-structured, deterministic tree model quite different from the relational model we use here. We are glossing over many details here; characterizing the precise relationship between these approaches (and other recent proposals for data provenance in queries [7] and updates [3, 2]) is beyond the scope of this article.

Now we consider a second example, a query Q_4 with grouping and aggregation:

```
SELECT Name, AVERAGE(Salary)
FROM Employees
WHERE Year >= 2005
GROUP BY Name
```

This query returns the names and average salaries since 2005 of all employees; a sample input database and result is shown in Figure 5. Note that Alice has no entries since 2004 so does not appear in the result.

In the previous example, we considered only dependences of output fields on input fields; the relation and record dependences are not very interesting for this example. Relation and record dependences become more important for queries such as Q_4 involving grouping and aggregation.

Figures 6(a–c) show the data slices for the whole output, record u_1 (and u_1.Salary), and field u_1.Name, respectively. The whole output depends on everything in the input except for Alice's salary fields; changing them cannot affect the output, but other changes may. The dependency provenance of u_1 is shown in Figure 6(b). The presence of record u_1 clearly depends on all of the data in t_2 and t_3; changing any of these fields may affect the average, which would replace u_1 with some other record (James, avg$'$). Record u_1 also depends on t_1.Year and t_6.Name. The reason is that changing '2004' to '2008' in t_1 or changing 'Bob' to 'James' in t_6 would affect the average associated with James in the output. Coincidentally, the provenance of u_1.Salary turns out to be the same as the provenance of u_1, and the reasoning is similar. Finally, in Figure 6(c), we see that u_1.Name does not (directly) depend on anything in the input. Of course, the presence of u_1 does depends on several parts of the input, so u_1.Name depends "indirectly" on these parts as well, but there is no single field in the input that we can change that will change u_1.Name in the result.

Employees			
id	Name	Salary	Year
t_1	James	1000	2004
t_2	James	1100	2005
t_3	James	1200	2006
t_4	Alice	1900	2003
t_5	Alice	2000	2004
t_6	Bob	1000	2006

$\Longrightarrow$

id	Name	Salary
u_1	James	1150
u_2	Bob	1000

Figure 5: Input data and result of running query Q_4

id	Name	Salary	Year
t_1	James	1000	2004
t_2	James	1100	2005
t_3	James	1200	2006
t_4	Alice	$\cdots$	2003
t_5	Alice	$\cdots$	2004
t_6	Bob	1000	2006

(a) For whole output

id	Name	Salary	Year
t_1	$\cdots$	$\cdots$	2004
t_2	James	1100	2005
t_3	James	1200	2006
t_6	Bob	$\cdots$	$\cdots$
$\vdots$			

(b) For u_1 and u_1.Salary

id	Name	Salary	Year
$\vdots$			

(c) For u_1.Name

Figure 6: Data slices for Q_4

4 Conclusions

We believe that the key question any approach to provenance must answer is what the provenance information *explains* about a query result in the context of the input data and query semantics that is not conveyed by the query result value itself. Previous approaches, such as why-provenance, where-provenance, and lineage have been based on intuitive notions of explanations such as identifying the source data that "influenced" or "contributed to" a part of the output or from which a part of the output was "copied". However, corresponding semantic correctness properties relating these forms of provenance to the actual semantics of a query have proven elusive or hard to generalize beyond monotone queries.

We have outlined one approach, dependency provenance, which is based on well-understood techniques from programming languages such as dependency analysis and program slicing. We believe this approach captures intuitions similar to those motivating other provenance techniques, but may be easier to generalize to the full range of features found in databases, including grouping, aggregation and stored procedures. However, this work is still relatively speculative and more research is needed to determine the feasibility of computing (or conservatively approximating) dependency provenance in practice and scale. Nevertheless, there appears to be a deep connection between program slicing and data provenance that we may be able to exploit by transferring ideas, tools, and techniques from programming languages research.

References

[1] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *POPL*, pages 147–160, New York, NY, USA, 1999. ACM Press.

[2] Peter Buneman, Adriane P. Chapman, and James Cheney. Provenance management in curated databases. In *Proceedings of the 2006 SIGMOD Conference on Management of Data*, pages 539–550, Chicago, IL, 2006. ACM Press.

[3] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. In *ICDT 2007*, number 4353 in Lecture Notes in Computer Science, pages 209–223. Springer, 2007.

[4] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *Proc. 2001 International Conference on Database Theory*, number 1973 in LNCS, pages 316–330. Springer-Verlag, 2001.

[5] James Cheney, Amal Ahmed, and Umut A. Acar. Provenance as dependency analysis. In M. Arenas and M. I. Schwartzbach, editors, *Proceedings of the 11th International Symposium on Database Programming Languages (DBPL 2007)*, number 4797 in LNCS, pages 139–153. Springer-Verlag, 2007.

[6] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[7] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, New York, NY, USA, 2007. ACM Press.

[8] Erhard Rahm and Hong-Hai Do. Data cleaning: Problems and current approaches. *IEEE Bulletin of the Technical Committee on Data Engineering*, 23(4), December 2000.

[9] Wang-Chiew Tan. Provenance in databases: Past, current, future. This issue.

[10] F. Tip. A survey of program slicing techniques. *Journal of programming languages*, 3:121–189, 1995.

[11] Mark Weiser. Program slicing. In *ICSE*, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.

[12] Mark Weiser. Programmers use slices when debugging. *Commun. ACM*, 25(7):446–452, 1982.

Recording Provenance for SQL Queries and Updates

Stijn Vansummeren[*]
Hasselt University and
Transnational University of Limburg, Belgium

James Cheney
University of Edinburgh
UK

Abstract

Knowing the origin of data (i.e., where the data was copied or created from)—its provenance*—is vital for assessing the trustworthiness of contemporary scientific databases such as UniProt [16] and SWISS-PROT [14]. Unfortunately, provenance information must currently be recorded manually, by added effort of the database maintainer. Since such maintenance is tedious and error-prone, it is desirable to provide support for recording provenance in the database system itself. We review a recent proposal for incorporating such support, as well as its theoretical properties.*

1 Introduction

Chris, a fan of foreign and domestic beers, constructs a database $R(\mathsf{beer}, \mathsf{kind}, \mathsf{origin})$ listing beers, their kind, and their origin. He proceeds by manually inserting tuples, as well as by copying from the existing general beer database $S(\mathsf{beer}, \mathsf{kind}, \mathsf{origin})$, and from $T(\mathsf{beer}, \mathsf{origin})$, a database that specializes in lagers.

$$\text{insert into } R \text{ values ('Duvel', 'Strong ale', 'Belgium');} \tag{1}$$
$$\text{insert into } R \text{ (select } * \text{ from } S \text{ where origin} = \text{'USA');} \tag{2}$$
$$\text{insert into } R \text{ (select T.beer, 'Lager' as kind, } T\text{.origin from } T); \tag{3}$$

When inspecting the result, Chris notices that R reports Stella Artois as an American beer, while it is in fact a Belgian one. A friend tells Chris that this error is probably due to database T, which is known for its poor data quality. Perhaps Chris should check the other records inserted from T for their correctness?

Although the scenario above is clearly a simplification for the purpose of illustration, many contemporary scientific databases—sometimes referred to as *curated databases*—are constructed in a similar manner by a labor-intensive process of copying, correcting, and annotating data from other sources. The value of curated databases lies in their organization and in the trustworthiness of their data. As illustrated above, knowing where data was copied or created from—its *provenance*—is particularly important in assessing the latter.

In hindsight, Chris could simply have recorded provenance by adding an extra attribute prov to R and by issuing slightly different update statements. For instance, update (3) would have become

$$\text{insert into } R \text{ (select } T\text{.beer, 'Lager' as kind, } T\text{.origin, 'db T' as prov from } T). \tag{4}$$

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[*]Stijn Vansummeren is a Postdoctoral Fellow of the Research Foundation–Flanders (FWO).

This only records provenance of whole tuples, however, and a finer granularity is often required. For instance, the presence of (Stella Artois, lager, USA, db T) in R would designate T as the provenance of the tuple (Stella Artois, lager, USA) even though this tuple does not literally occur in T. After all, only Stella Artois and USA were copied from T, but lager was inserted by Chris. On the other hand, recording only the provenance of data values (like Stella Artois, lager, ...) is often not sufficient either. For instance, knowing that all data values in (McChouffe, Scottish ale, Belgium) were copied from S does not necessarily imply that all of these data values came from the same record in S. As such, it is desirable to record provenance at all levels in a database (data values, tuples, and even whole tables). While it is possible to do so manually by adding enough extra attributes to R and by suitably rewriting the original updates (1), (2), and (3), this approach quickly becomes very tedious, time-consuming, and error-prone, especially when R contains many attributes. Also, the importance of retaining detailed provenance information is often not appreciated until it is too late—perhaps months or years after the data was originally copied into the database. In this respect, it seems preferable to let the user write queries and updates as before, and to let the database system record provenance *automatically*.

Before provenance recording can be automated, however, it is imperative to have a good explanation of the meaning of queries and updates with regard to provenance. This meaning may be obvious for the examples given so far, but what about updates such as the following?

$$\textsf{insert into } R \textsf{ (select } S\textsf{.beer, 'stout' as kind, } S\textsf{.origin from } S \textsf{ where } S\textsf{.kind} = \textsf{'stout')} \quad (5)$$

Is stout created by Chris or copied from S? Both explanations seem reasonable due to the condition that S.kind = 'stout'. A similar situation occurs for updates involving joins:

$$\textsf{insert into } R \textsf{ (select } S\textsf{.beer, } S\textsf{.kind, } T\textsf{.origin from } S, T \textsf{ where } S\textsf{.beer} = T\textsf{.beer and } S\textsf{.origin} = T\textsf{.origin;)} \quad (6)$$

Are the beer and origin attributes copied from S or from T? Again, both explanations seem reasonable, but it is unclear which explanation is to be preferred.

In this article, we will follow Wang and Madnick [17] and Bhagwat et al. [2] and define stout to be created by Chris in update (5) because stout appears as a constant in the select clause instead of S.kind. Moreover, we define beer to be copied from S in update (6) because the select clause lists S.beer and not T.beer. Similarly, origin is taken to be copied from T because the select clause lists T.origin and not S.origin.

While this provenance semantics is simple and natural, it may not be the particular provenance that a database curator had in mind for the above updates. Nevertheless, this simple provenance semantics has been shown *expressively complete*: for every query or update O that manually records provenance (like (4) above) there exists a normal query or update (like (1), (2), and (3) above) for which the provenance semantics is equivalent to O, provided that O satisfies certain soundness criteria discussed in Sections 2 and 3. As such, if we use this semantics to record provenance automatically, we do not lose flexibility with regard to recording provenance manually. We feel that this property strongly argues in favor of the proposed provenance semantics as the "right" basis for recording provenance automatically.

We should note that although we will restrict ourselves in what follows to the provenance semantics for (a fragment of) SQL queries and updates operating on classical flat relations and only consider provenance at the data value and tuple level, the topic was originally studied for queries and updates operating on *nested* relations, where provenance is recorded at all levels (data values, tuples, and tables) [5]. We refer the interested reader to Buneman et al. [6] for a full exposition.

To put this article in the right context, we should also mention the other forms of provenance recently studied in databases. First, while we are interested in recording *where* data is copied or created from, the *why*-provenance approach of Cui et al. [6] and Buneman et al. [4] wishes to identify, for each output tuple t of a query, the set of input tuples that caused t to be output. More recently, why-provenance has been refined using program slicing techniques by Cheney et al. [8]. The *how*-provenance approach of Green et al. [7] is interested in recording how t was computed from the input (e.g., t could be the result of joining two input tuples). Finally, there has also been interest on *querying* provenance and other forms of annotations rather than recording it [12, 11].

beer	origin	beer^c	origin^c	tup^c
Leinenkugel	USA	c_1	c_2	c_3
Stella Artois	USA	c_4	c_5	c_6

beer	origin	beer^c	origin^c	tup^c
Leinenkugel	USA	c_1	c_2	c_3
Stella Artois	Belgium	c_4	$\perp$	$\perp$

Figure 1: Color propagation for query (7). On the left is $\mathsf{clr}(T)$, the colored version of table T, and on the right is the colored result.

2 Provenance Recording for Queries

Let us first consider provenance recording for queries. Updates will be considered in Section 3. For ease of exposition we restrict ourselves to simple SQL queries of the following form, excluding subqueries; grouping; and aggregation.

$$\begin{array}{rcl} Q & ::= & \mathsf{select}\ r_i.* \ \mathsf{from}\ R_1\ r_1,\ \ldots,\ R_m\ r_m\ \mathsf{where}\ \varphi \\ & | & \mathsf{select}\ a_1\ \mathsf{as}\ A_1,\ \ldots,\ a_n\ \mathsf{as}\ A_n\ \mathsf{from}\ R_1\ r_1,\ \ldots,\ R_m\ r_m\ \mathsf{where}\ \varphi \\ & | & Q\ \mathsf{union}\ Q \end{array}$$

Here, φ is any valid where-clause without subqueries and every a_i is either a constant data value or an expression such as $r_i.C$ that refers to an attribute of one of the tuple variables. Our approach can be generalized to deal with subqueries, grouping, and the connectives intersect and except, but aggregation presents some problems, as we will see. Note that we only allow the wildcard $*$ to be applied to a single tuple variable; selections such as select $*$ from $R,\ S$ that return all attributes of a cartesian product can always be rewritten to mention these attributes explicitly in the select clause.

The provenance semantics. Let us collectively refer to the individual data values and tuples in a database as the database's *items*. To define the provenance semantics, we use a formalization based on the "tagging" or "annotation propagation" approach of Wang and Madnick [17] and Bhagwat et al. [2]. In this approach, each input item is assumed to have an identifying "color" which serves as an abstraction of a system identifier or some other means of referring to part of a database. We can then describe how queries and updates manipulate provenance by means of functions mapping such colored databases to colored databases in which colors are propagated along with their item during computation of the output. The provenance of an item in the output is simply the item in the input with the same color. To illustrate, consider the table $T(\mathsf{beer}, \mathsf{origin})$ from the Introduction with tuples $\{(\mathsf{Leinenkugel}, \mathsf{USA}), (\mathsf{Stella\ Artois}, \mathsf{USA})\}$ in which the data values and the tuples are annotated with colors $c_1, c_2, \ldots$ as shown at the left of Fig. 1. There, beer^c and origin^c store the colors of the data values in the beer and origin attributes, and tup^c stores the colors of the tuples. As such, Leinenkugel is colored by c_1, the first occurrence of USA is colored by c_2, the first tuple is colored by c_3, and so on. We could then define the colored semantics of the SQL query

$$\begin{array}{c} (\mathsf{select\ t.*\ from}\ T\ \mathsf{t\ where\ t.beer <> 'Stella\ Artois'}) \\ \mathsf{union\ (select\ t.beer,\ 'Belgium'\ as\ origin\ from}\ T\ \mathsf{t\ where\ t.beer = 'Stella\ Artois'}) \end{array} \tag{7}$$

to map T to the colored table at the right of Fig. 1. This defines the provenance of Leinenkugel in the output to be the corresponding data value in T, the provenance of the tuple $(\mathsf{Leinenkugel}, \mathsf{USA})$ to be the provenance of the first tuple in T, and so on. The "empty" or "blank" color $\perp$ indicates that an item is introduced by the query itself. Hence, this particular colored semantics takes the view that the second select subquery constructs a new tuple rather than copying an existing one.

Intuitively, we will take the view that queries either construct new items or copy complete items from the input. As such, all data values resulting from constant construction as in select 'USA' as origin from $T\ t$ are colored $\perp$, as are the tuples returned by queries such as select $A,\ B$ from R whose select clause constructs new tuples. All other items, such as the tuples returned by select $\mathsf{t.*}$ from $T\ t$, retain their color. This is essentially the same provenance semantics as that of Wang and Madnick [17] and Bhagwat et al. [2], although they only consider provenance for data values, not tuples.

In order to elegantly formalize this intuition, notice that, by storing colors as in Fig. 1, it becomes possible to define functions mapping colored databases to colored tables in SQL itself. For example, if we let $\mathsf{clr}(T)$ denote the colored version of table $T(\mathsf{beer}, \mathsf{origin})$ then the particular colored semantics of query (7) illustrated in Fig. 1 is defined by

$$\begin{aligned} &(\mathsf{select\ t.*\ from\ clr}(T)\ \mathsf{t\ where\ t.beer} <> \text{'Stella Artois'}) \\ \mathsf{union}\ &(\mathsf{select\ t.beer\ as\ beer},\ \text{'Belgium'}\ \mathsf{as\ origin},\ \mathsf{t.beer^c\ as\ beer^c},\ \perp \mathsf{as\ origin}^c,\ \perp \mathsf{as\ tup}^c \\ &\ \mathsf{from\ clr}(T)\ \mathsf{t\ where\ t.beer} = \text{'Stella Artois'}) \end{aligned} \tag{8}$$

To define our provenance semantics it hence suffices to assign, to each query Q, a query $\mathcal{P}[Q]$ mapping colored databases to colored tables. It is important to remark that Q and $\mathcal{P}[Q]$ operate on different views of the database: Q operates on the tables without colors (like (7) above), while $\mathcal{P}[Q]$ operates on colored tables (like (8)). To avoid confusion between the two views, we will range over uncolored tables by R, S, and T, and over their colored versions by $\mathsf{clr}(R), \mathsf{clr}(S)$, and $\mathsf{clr}(T)$. We refer to the attributes that store normal data values in $\mathsf{clr}(R), \mathsf{clr}(S)$, and $\mathsf{clr}(T)$ (like beer and origin) as the *normal attributes* and to the attributes that store colors (like $\mathsf{beer}^c, \mathsf{origin}^c$, and tup^c) as the *color attributes*.

Definition 1: The provenance semantics $\mathcal{P}[Q]$ of a query Q operating on uncolored tables is inductively defined as follows. Let $\mathcal{P}[a]$ denote the blank color $\perp$ when a is a constant, and let $\mathcal{P}[a]$ denote $t.A^c$ when a is an attribute reference $t.A$ with t a tuple variable.

- $\mathcal{P}[\mathsf{select}\ r_i.*\ \mathsf{from}\ R_1\ r_1,\ \ldots,\ R_m\ r_m\ \mathsf{where}\ \varphi] :=$
$$\mathsf{select}\ r_i.*\ \mathsf{from\ clr}(R_1)\ r_1,\ \ldots,\ \mathsf{clr}(R_m)\ r_m\ \mathsf{where}\ \varphi;$$
- $\mathcal{P}[\mathsf{select}\ a_1\ \mathsf{as}\ A_1,\ \ldots,\ a_n\ \mathsf{as}\ A_n\ \mathsf{from}\ R_1\ r_1,\ \ldots,\ R_m\ r_m\ \mathsf{where}\ \varphi] :=$
$$\begin{aligned} &\mathsf{select}\ a_1\ \mathsf{as}\ A_1,\ \ldots,\ a_n\ \mathsf{as}\ A_n,\ \mathcal{P}[a_1]\ \mathsf{as}\ A_1^c,\ \ldots,\ \mathcal{P}[a_n]\ \mathsf{as}\ A_n^c,\ \perp \mathsf{as\ tup}^c \\ &\mathsf{from\ clr}(R_1)\ r_1,\ \ldots,\ \mathsf{clr}(R_m)\ r_m\ \mathsf{where}\ \varphi; \end{aligned}$$
- $\mathcal{P}[Q_1\ \mathsf{union}\ Q_2] := \mathcal{P}[Q_1]\ \mathsf{union}\ \mathcal{P}[Q_2]$.

Example 1: To illustrate, $\mathcal{P}[Q]$ with $Q = \mathsf{select}\ s.\mathsf{beer},\ \text{'stout'}\ \mathsf{as\ kind},\ s.\mathsf{origin\ from}\ S\ s\ \mathsf{where}\ s.\mathsf{kind} = \text{'stout'}$ as in example (5) from the Introduction yields

$$\begin{aligned} &\mathsf{select\ s.beer},\ \text{'stout'}\ \mathsf{as\ kind, s.origin, s.beer^c\ as\ beer^c},\ \perp \mathsf{as\ kind}^c,\ \mathsf{s.origin^c\ as\ origin^c},\ \perp \mathsf{as\ tup}^c \\ &\mathsf{from\ clr}(S)\ \mathsf{s\ where\ s.kind} = \text{'stout'}. \end{aligned} \tag{9}$$

Also, $\mathcal{P}[Q]$ with Q as in query (7) yields query (8).

Inherent to definition of $\mathcal{P}[Q]$ is that queries that are equivalent under the normal semantics need not be equivalent under the provenance semantics. For example $Q_1 := \mathsf{select}\ r.*\ \mathsf{from}\ R\ r$ is equivalent to $Q_2 := \mathsf{select}\ r.A\ \mathsf{as}\ A, r.B\ \mathsf{as}\ B\ \mathsf{from}\ R\ r$ when R consists only of attributes A and B, but $\mathcal{P}[Q_1]$ is not equivalent to $\mathcal{P}[Q_2]$ as the former retains the tuple colors from the input, while the latter colors all tuples $\perp$.

Expressive completeness Let us now see how this provenance semantics compares with the manual approach to recording provenance. In this respect, note that queries mapping colored databases to colored tables, such as (8) above, can also be viewed as being *manually constructed* to record provenance. In other words, we want to compare the class of queries $\{\mathcal{P}[Q] \mid Q$ a query on uncolored databases$\}$ with the class of queries P mapping colored databases to colored tables. Since we are interested in *recording* provenance, however, we will exclude from our discussion queries P such as

$$\mathsf{select\ t.}{*}\ \mathsf{from\ clr}(T)\ \mathsf{t\ where\ t.beer}^{c} = c_1 \tag{10}$$

that *query* provnenance rather than record it.

Definition 2: A *provenance recording* query is a query P mapping colored databases to colored tables in which every where-clause φ mentions only normal attributes.

Clearly, (10) is hence not provenance recording. In contrast, $\mathcal{P}[Q]$ is always provenance-recording since the where-clause of a query Q operating on uncolored tables only mentions normal attributes and since $\mathcal{P}[Q]$ does not affect where-clauses.

Can every provenance-recording query be defined in terms of $\mathcal{P}[Q]$ for some Q? The answer is no, for two reasons. First, due to our view of queries as either constructing new items or copying whole items, it is impossible for $\mathcal{P}[Q]$ to yield something like

$$\mathsf{select\ t.beer\ as\ beer},\ \mathsf{t.beer}^{c}\ \mathsf{as\ beer}^{c},\ \mathsf{t.tup}^{c}\ \mathsf{as\ tup}^{c}\ \mathsf{from\ clr}(T)\ \mathsf{t} \tag{11}$$

that returns tuples which do not literally occur in T, yet have the same colors as tuples in T. Similarly, it is impossible for $\mathcal{P}[Q]$ to yield something like

$$\mathsf{select\ 'USA'\ as\ origin},\ \mathsf{t.origin^c\ as\ origin}^{c},\ \perp\ \mathsf{as\ tup}^{c}\ \mathsf{from\ clr}(T)\ \mathsf{t} \tag{12}$$

in which data values are given the provenance of data values from T although the data value itself need not occur in T. We refer to provenance recording queries that only color output items i by color c if i also occurs in the input as '*copying*'. (See [5, 6] for a full formal definition of this concept).

Second, due to our view that only data values constructed by a constant expression are colored $\perp$, it is impossible for $\mathcal{P}[Q]$ to yield something like

$$\mathsf{select\ t.beer\ as\ beer},\ \perp\ \mathsf{as\ beer}^{c},\ \perp\ \mathsf{as\ tup}^{c}\ \mathsf{from\ clr}(T)\ \mathsf{t} \tag{13}$$

that colors every possible beer data value by $\perp$. Indeed, to simulate (13) by means of $\mathcal{P}[Q]$, Q would have to mention every possible beer value as a constant, of which there are unboundedly many. We call queries that can color only a finite, bounded number of atoms by $\perp$ '*bounded inventing*'.

We view the fact that $\mathcal{P}[Q]$ can only define provenance recording queries that are 'copying' and 'bounded inventing' as a desirable property: it ensures that the provenance relationship between input and output described by $\mathcal{P}[Q]$ is not arbitrary, but meaningful. After all, one could hardly argue that the provenance relationships described by (11) and (12) above correspond to the intuitive notion "is copied from T". Similarly, queries without aggregation are typically considered as "domain-preserving" in database theory, with limited ability to create new data values. The bounded inventing property merely ensures that the provenance semantics respects this view. With regard to the copying and bounded inventing queries, our provenance semantics can be shown expressively complete:

Proposition 3 (Buneman et al. [5, 6]): For every provenance recording query P that is copying and bounded inventing there exists a query Q mapping uncolored databases to uncolored tables such that $P \equiv \mathcal{P}[Q]$.

As such, we are ensured that we do not lose flexibility when using $\mathcal{P}[Q]$ to record provenance automatically instead of recording provenance manually. Of course, in practice we also would like to record provenance for queries involving aggregation such as select $A, \mathsf{sum}(B)$ from R group by A that fall outside the current framework. Indeed, although we could in principle simply define the provenance of all atomic data values resulting from sum to be created by the query itself, this causes the provenance semantics to become *un*bounded inventing. A more satisfying approach than simply defining the results of an aggregation operator to be created by the query itself could be to record *how* this result was computed. For example, we could color a data value resulting from the above sum aggregation by $\mathsf{sum}(c_1, c_2, c_3)$ indicating that it was obtained by applying sum to the set of data values from the input colored by c_1, c_2, and c_3, respectively. This use of *expressions* as provenance is similar to the approach of Green et al. [7], who use semi-ring expressions to describe the provenance of relational algebra queries without aggregation. It is also analogous to certain techniques for *workflow* provenance, as known from the geospatial and Grid computing communities [3, 10, 15]. It is not clear, however, whether and how our expressive completeness results transfer to this setting.

3 Provenance Recording for Updates

Our discussion of provenance for queries is straightforwardly extended to updates like (2) and (3) from the Introduction of the form insert into $R\ Q$: the provenance of the inserted items is simply given by $\mathcal{P}[Q]$.

Definition 4 (Provenance of query insertion): $\mathcal{P}[\mathsf{insert\ into}\ R\ Q] := \mathsf{insert\ into\ clr}(R)\ \mathcal{P}[Q]$.

For instance, for update (3) from the Introduction this yields

$$\begin{aligned}&\mathsf{insert\ into\ clr}(R)\ \mathsf{select\ t.beer,\ 'Lager'\ as\ kind,\ t.origin,}\\ &\qquad\qquad\qquad \mathsf{t.beer}^c\ \mathsf{as\ beer}^c,\ \perp\ \mathsf{as\ kind}^c, \mathsf{t.origin}^c\ \mathsf{as\ origin}^c, \perp \mathsf{as\ tup}^c\ \mathsf{from\ clr}(T)\ \mathsf{t}.\end{aligned} \tag{14}$$

Updates of the form insert into $R(A, \ldots, B)$ values $(d, \ldots, d')$ like (1) from the Introduction clearly add newly constructed items to R. We hence define:

Definition 5 (Provenance of value insertion):

$$\begin{aligned}\mathcal{P}[\mathsf{insert\ into}\ R(A,\ldots,B)\ \mathsf{values}\ (d,\ldots,d')] :=& \mathsf{insert\ into\ clr}(R)(A,\ldots,B,A^c,\ldots,B^c,\mathsf{tup}^c)\\ & \mathsf{values}\ (d,\ldots,d',\perp,\ldots,\perp),\end{aligned}$$

where we assume for ease of exposition that $A, \ldots, B$ comprise all attributes of R.

The provenance semantics of delete statements is also straight-forward, as deleting a tuple also deletes its provenance.

Definition 6 (Provenance of deletion): $\mathcal{P}[\mathsf{delete\ from}\ R\ \mathsf{where}\ \varphi] := \mathsf{delete\ from\ clr}(R)\ \mathsf{where}\ \varphi$.

Observe that for the updates considered so far, $\mathcal{P}[U]$ is still 'copying' and 'bounded inventing'. Moreover, the provenance semantics is still expressively complete with regard to the class of updates from colored databases to colored databases that manually record provenance, in the following sense. Similar to the case for queries, we exclude from our discussion updates such as delete from $\mathsf{clr}(T)$ t where $\mathsf{t.beer}^c = c_1$ that *query* provenance rather than record it.

Definition 7: A *provenance recording* update is an update mapping colored databases to colored tables in which every where-clause φ mentions only normal attributes.

beer	origin	beer^c	origin^c	tup^c
Leinenkugel	USA	c_1	c_2	c_3
Stella Artois	USA	c_4	c_5	c_6

beer	origin	beer^c	origin^c	tup^c
Leinenkugel	USA	c_1	c_2	c_3
Stella Artois	Belgium	c_4	$\perp$	c_6

Figure 2: Color propagation for update (15). On the left is $\mathsf{clr}(T)$, the colored version of table T, and on the right is the colored result.

Proposition 8: Let V be a provenance recording update of the following form.

$$U \quad ::= \quad \mathsf{insert\ into}\ R\ Q \mid \mathsf{insert\ into}\ R(A,\dots,B)\ \mathsf{values}\ (d,\dots,d') \mid \mathsf{delete\ from}\ R\ \mathsf{where}\ \varphi.$$

If V is copying and bounded inventing, then there exists an update U operating on uncolored databases, also of the above form, such that $V \equiv \mathcal{P}[U]$.

Let us now consider update statements. In this respect, note that updates such as the following intuitively do not construct new tuples but modify existing ones "in-place".

$$\mathsf{update}\ T\ \mathsf{set\ origin} = \text{'}\mathsf{Belgium}\text{'}\ \mathsf{where\ beer} = \text{'}\mathsf{Stella\ Artois}\text{'} \tag{15}$$

It hence seems reasonable to define their provenance semantics in a way that agrees with how system identifiers are preserved in practical database management systems.

Definition 9 (Provenance of updates): $\mathcal{P}[\mathsf{update}\ R\ \mathsf{set}\ (A,\dots,B) = (d,\dots,e)\ \mathsf{where}\ \varphi] := \mathsf{update\ clr}(R)$ $\mathsf{set}\ (A,\dots,B,A^c,\dots,B^c) = (d,\dots,e,\perp,\dots,\perp)\ \mathsf{where}\ \varphi$.

For instance, for update (15) above this yields

$$\mathsf{update\ clr}(T)\ \mathsf{set}\ (\mathsf{origin},\mathsf{origin}^c) = (\text{'}\mathsf{Belgium}\text{'},\ \perp)\ \mathsf{where\ beer} = \text{'}\mathsf{Stella\ Artois}\text{'}. \tag{16}$$

Note that although update (15) and query (7) essentially express the same database transformation on uncolored tables, their provenance semantics differs significantly. Indeed the query maps the colored table at the left of Fig. 1 to the colored table at the right of Fig. 1, while the update maps that same table to the colored table at the right of Fig. 2. In particular, the provenance semantics of the update is *not* copying, as the first output tuple is not identical to the first input tuple, although they are colored the same. The provenance semantics of the update statement is '*kind preserving*' however: it will only color an output atom by color c if the atom also occurs in the input with color c; and it will only color an output tuple by color c if there is a tuple in the input with color c. We refer to Buneman et al. [5, 6] for a full definition of this concept.

Every copying V is also kind preserving. As such, the provenance semantics $\mathcal{P}[U]$ of all updates U considered in this article is kind preserving. The provenance semantics is *not* expressively complete with regard to the class of kind preserving and bounded inventing provenance recording updates, however. To see why, suppose that R consists only of the attribute origin, and further suppose that we want to simulate the update

$$\mathsf{insert\ into\ clr}(R)\ (\mathsf{select}\ \text{'}\mathsf{Belgium}\text{'}\ \mathsf{as\ origin},\ \mathsf{t.origin^c\ as\ origin}^c,\ \perp\ \mathsf{as\ tup}^c\ \mathsf{from\ clr}(T)\ \mathsf{t}), \tag{17}$$

which is kind preserving, but not copying. (The inserted tuples are colored the same as tuples of T, but are not identical.) To simulate this update in terms of $\mathcal{P}[U]$ for some U, U clearly needs to be an insert statement itself. We already know, however, that this implies that $\mathcal{P}[U]$ is copying; as such it cannot express (17). Nevertheless, by adding extra update operators it is possible to regain expressive completeness; see Buneman et al [5, 6].

4 Conclusion

In order to assess the trustworthiness of a database it is vital to know the provenance of its data. Since manually recording such provenance quickly becomes very tedious, time-consuming, and error-prone, is preferable to let the user write queries and updates as before, and to let the database system record provenance automatically. In this respect, it is imperative to have a good explanation of the meaning of queries and updates with regard to provenance. Fortunately, the intuitive view of queries as either constructing new items or copying whole items from the input, as well as the intuitive view of updates as modifying items in-place, yields an automatic provenance recording semantics that is guaranteed to be as flexible as recording provenance manually. We conclude this article by remarking that, while the full provenance semantics presented here remains to be implemented in practice, preliminary experiments by Bhagwat et al. [2] and Buneman et al. [4] suggest that the overhead incurred by recording provenance as opposed to not recording it is reasonable.

Acknowledgement We are grateful to Peter Buneman for introducing us to provenance in databases, and for the very enjoyable collaboration that led to the expressive completeness results presented here.

References

[1] Marcelo Arenas and Michael I. Schwartzbach, editors. *Database Programming Languages, 11th International Symposium, DBPL 2007, Vienna, Austria, September 23-24, 2007, Revised Selected Papers*, volume 4797 of *Lecture Notes in Computer Science*. Springer, 2007.

[2] Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. *VLDB Journal*, 14(4):373–396, 2005.

[3] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, 2005.

[4] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *SIGMOD 2006: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 539–550, Chicago, IL, 2006. ACM.

[5] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. In Thomas Schwentick and Dan Suciu, editors, *ICDT 2007: Proceedings of the 11th International Conference on Database Theory*, volume 4353 of *Lecture Notes in Computer Science*, pages 209–223, Barcelona, Spain, 2007. Springer.

[6] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. Technical report, 2007.

[7] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *ICDT 2001: Proceedings of the 8th International Conference, on Database Theory*, volume 1973 of *LNCS*, pages 316–330, London, UK, 2001. Springer.

[8] James Cheney, Amal Ahmed, and Umut A. Acar. Provenance as dependency analysis. In Arenas and Schwartzbach [1], pages 138–152.

[9] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[10] Ian Foster and Luc Moreau, editors. *Proceedings of the 2006 International Provenance and Annotation Workshop (IPAW 2006)*, number 4145 in LNCS. Springer-Verlag, 2006.

[11] Floris Geerts and Jan Van den Bussche. Relational completeness of query languages for annotated databases. In Arenas and Schwartzbach [1], pages 127–137.

[12] Floris Geerts, Anastasios Kementsietsidis, and Diego Milano. MONDRIAN: Annotating and querying databases through colors and blocks. In *ICDE 2006: Proceedings of the 22nd International Conference on Data Engineering*, page 82, Atlanta, Georgia, 2006. IEEE Computer Society.

[13] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS 2007: Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40, New York, NY, USA, 2007. ACM Press.

[14] European Molecular Biology Laboratory. Swiss-prot database.

[15] Yogesh Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[16] Universal Protein Resource. `http://www.ebi.uniprot.org/`.

[17] Y. Richard Wang and Stuart E. Madnick. A polygen model for heterogeneous database systems: The source tagging perspective. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 519–538, Brisbane, Queensland, Australia, 1990. Morgan Kaufmann.

Issues in Building Practical Provenance Systems

Adriane Chapman and H.V. Jagadish
University of Michigan
Ann Arbor, MI 48109
{apchapma, jag}@umich.edu

Abstract

The importance of maintaining provenance has been widely recognized, particularly with respect to highly-manipulated data. However, there are few deployed databases that provide provenance information with their data. We have constructed a database of protein interactions (MiMI), which is heavily used by biomedical scientists, by manipulating and integrating data from several popular biological sources. The provenance stored provides key information for assisting researchers in understanding and trusting the data. In this paper, we describe several desiderata for a practical provenance system, based on our experience from this system. We discuss the challenges that these requirements present, and outline solutions to several of these challenges that we have implemented. Our list of a dozen or so desiderata includes: efficiently capturing provenance from external applications; managing provenance size; and presenting provenance in a usable way. For example, data is often manipulated via provenance-unaware processes, but the associated provenance must still be tracked and stored. Additionally, provenance information can grow to outrageous proportions if it is either very rich or fine-grained, or both. Finally, when users view provenance data, they can usually understand a SELECT manipulation, but "why did the bcgCoalesce [1] manipulation output that?"

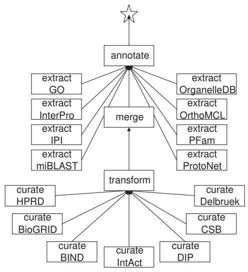
1 Introduction

Once upon a time, there lived a beautiful (and highly intelligent) researcher. She had a sad life chained to her lab bench day and night, slaving for her evil Principal Investigator, collecting data and analyzing numbers. One day a handsome Computer Scientist heard of the researcher's plight and decided to save the damsel in distress. First he built a program that would measure signal intensity in her experiments. Many more programs followed, each designed to reduce the tasks performed by the beautiful (and highly intelligent) researcher. The handsome Computer Scientist dazzled the evil Principal Investigator with the power of his programs and rescued the fair researcher from her lab bench. Just as they were about to ride into the sunset, the evil Principal Investigator popped her warty green face out of the tower and said, "I think you better come back in, I don't understand how or where you got these numbers, but they certainly can't be correct."

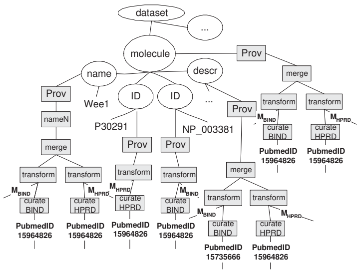
The handsome Computer Scientist laughed and cried, "Slaying this dragon will be easy. I maintained provenance!" Unfortunately, the provenance the handsome Computer Scientist kept was coarse-grained and not easy

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[0]Supported in part by NSF grant number IIS 0741620 and by NIH grant number U54 DA021519.



(a)

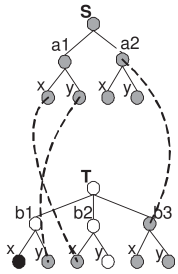


(b)

Figure 1: 1(a) The workflow used to generate MiMI. 1(b) A data item with provenance from MiMI.

```
Insert x into T/b2
Copy S/a1/y into T/b1/y
Insert y into T/b2
Copy S/a2 into T/b3
Copy S/a1/x into T/b2/x
```

(a)



(b)

Figure 2: 2(a) The user's actions on S and T. 2(b) The provenance links from T to S. Nodes originally in T are white; inserted nodes are black and copied nodes are grey to distinguish user actions.

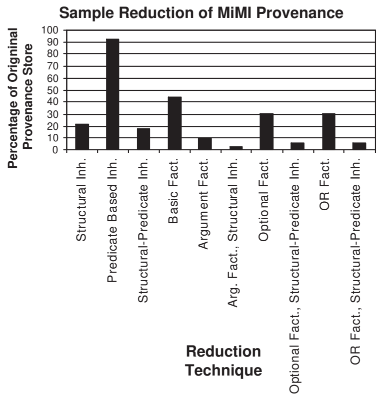


Figure 3: The storage savings for a set of reduction techniques applied to MiMI.

to query with the data itself. The handsome Computer Scientist and beautiful (and highly intelligent) researcher spent the remainder of their lives toiling to understand what happened to the data. *The End.*

The moral of this bedtime story: Don't just maintain provenance, maintain *good* provenance. Knowing that we should store provenance information doesn't mean we actually can, or do, or do it correctly. Even outside of fairy tales, researchers and scientists still have difficulty understanding what happened to their data, particularly when the data is heavily manipulated.

We have constructed a database of protein interactions, MiMI [16], by manipulating and integrating data from several popular biological sources. Figure 1(a) contains the general workflow used to generate MiMI. As scientists used the data in MiMI, it became apparent that provenance was needed to assist them in understanding and trusting the data presented. A snapshot of provenance information captured in MiMI is shown in Figure 1(b). While watching researchers use provenance information, we realized that their provenance information needs more than just a simple capture-store-fileaway approach. In Section 2, we present both required and recommended features for a database system incorporating provenance information based upon our experience with MiMI. Section 3 describes current provenance systems in light of these desiderata. In Sections 4 and 5 we discuss practical implementation options and conclude.

2 Desiderata

In this section, we will outline a set of features, required and recommended, needed for a database to incorporate provenance.

2.1 How Much Provenance to Capture

I. Granularity Choice *(Required)* Allow provenance to be captured and stored at every granularity. Currently there are two main trends for attaching provenance information: coarse and fine grained. Many workflow systems that generate provenance records attach provenance information at the coarse-grained file level [2, 12, 13, 14, 15]. Other specialized systems attach provenance at the fine grain of attribute [4, 6, 7, 8, 11]. Often, however, systems need a mix of usage. For example, in MiMI, provenance is attached to files, data items and attributes, as shown in Figure 1(b). When attributes, files and data items are broken up or used out of context, provenance is especially important at every granularity.

II. Exact Execution Provenance *(Required)* Record the exact provenance for each specific data item, not just the general provenance for a "class" of items. For example, attributes and data items within files behave differently through a given workflow based on data/attribute type, content, etc. The workflow to generate MiMI is shown in Figure 1(a). If a scientist wishes to know where the Wee1 name attribute came from, pointing to the workflow used is not enlightening, since via the workflow, that attribute could have come from any number of external sources, e.g. BIND, HPRD, etc. Instead, we wish to know that the Wee1 name attribute came from BIND and HPRD, while the P30291 ID attribute came only from HPRD. Moreover, while the MiMI.xml *file* went through a merge process, the P30291 ID attribute never merged with any other information.

III. Provenance Information *(Required)* Permit variation of the form or content of the provenance information. Current provenance systems capture a huge range of information from information about the files used and produced and the scripts run [2, 12, 13, 14, 15] to user annotations [3, 19]. But what exactly is needed to allow individuals to utilize the data? In MiMI, we found storing a mix of provenance information the most successful. For instance, HPRD describes each protein in an XML file, and MiMI's provenance should reference the exact XML file used. On the other hand, user annotations, such as the PubMedID (a unique identifier for biology research articles) used to garner the original information should also be kept. In other words, a provenance system should be flexible enough to store a large range of information as determined by the application.

IV. Capturing Non-automated processes *(Required)* Provide the ability to capture manipulations that are performed outside of automated workflows. While capturing the exact execution for every file, data item and attribute, it is imperative not to miss the actions performed manually by a curator. For instance, in MiMI, because the identity functions that dictate which proteins to merge are generated automatically, an expert user will find a mistake occasionally. The manual correction of this mistake must be reflected in the provenance records. Automatic capture of workflows alone is not enough.

2.2 Systems Issues

V. Source Data Item Identity *(Required)* Keep track of your incoming data. No matter what information is ultimately retained in the data set or provenance store, there must always be a firm, unbending representation for data item identity. Consider the problem in MiMI: 232,680 proteins from seven sources are merged into 117,549 proteins. When the merge process takes place, how do you identify the original components and where they came from? How do you go backwards to look at the original proteins? Even specifying that a protein is from BIND is not enough, since several proteins from BIND can be merged into one. You cannot trace back any further without some notion of source data item identity.

VI. Provenance Storage Size *(Required)* Plan for large provenance store costs. Given the amount of provenance material stored, provenance stores can grow to immense sizes, and easily outstrip the size of the data. MiMI is 270MB; the associated provenance store is 6GB before compression.

VII. Manipulation Information *(Recommended)* Maintain detailed manipulation information. Most provenance systems keep track of the scripts or manipulations that have been applied to the data. Some, such as [14], allow users to modify process order, and change applications to achieve the desired results. However, this requires an innate knowledge of each process, such as *SELECT* or *bcgCoalese* [1]. An alternative approach would be to maintain information to generate result explanations. Thus, when a user asks, "Why did P30291 from HPRD merge with NP_003381 from BIND?" no innate knowledge of the merge process is required for the answer; it can be automatically derived based on information in the provenance store.

VIII. Inter-system Provenance *(Recommended)* Build toward inter-operability of provenance systems. As systems grow and become interconnected, provenance should be interchangeable. As MiMI has grown in popularity, it has become a reference to other applications such as PubViz [22]. These systems also attempt to maintain some notion of provenance. However, they should not be required to store provenance information found in MiMI. Instead, they should store the provenance associated with their actions, then point to MiMI for the provenance beyond their borders.

2.3 Usability

IX. User Interactions *(Required)* Allow users to actively utilize provenance information at many levels. As discussed previously, there can be a huge amount of provenance information to trawl through. This information should not be stored away out of sight in case there is major problem, it should be available to end users. How can an end user navigate this deluge of information? In MiMI, we have noticed that user's needs fall into several categories: dataset generation overview, data item overview and particular manipulation overview. Users should be able to see provenance information at many levels.

X. Provenance Queriability with the Data *(Required)* Provide support for querying provenance and data together. Provenance is an essential component in assisting end users in trusting and using the data. To this end, the provenance information should be queriable with the data itself. In MiMI, queries often consist of intersections of data and provenance. For instance, "Return all molecules located in the mitochondria *(data)* that were reported by HPRD or IntAct *(provenance)*." Making the provenance records available, but forcing users to do a processing step to join them with the data is an undue burden.

XI. Error Finding and Fixing *(Recommended)* Enable easy provenance store maintenance. Consider the following scenario: a user queries MiMI, and notices two molecules have been merged that should not have been. The user reports it. What happens? Hopefully the error will be corrected and the two mis-merged proteins will be separated. But what about the provenance store? Mechanisms must be in place to incrementally update the provenance store to allow for error finding and fixing.

3 Current Provenance Systems

There are several provenance systems that have been applied to real scientific data [5, 21], and espouse many of the desiderata discussed above. The PASOA project [15, 17] has been applied to several real-world scientific endeavors. It is concerned with the origins of a result or determining when results are invalid and has paid specific attention to desiderata VII, VIII and X. Chimera [12], is concerned with data derivation and shines in desiderata V, VI and VIII. Additionally, myGRID [13] is a collaborative environment for scientists with provenance handling; myGRID handles desiderata VIII and IX. Other workflow systems have integrated provenance information such as VisTrails [14], Redux [2], and those participating in the International Provenance and Annotation Workshop

Challenge [18]. In general these systems are thoughtful of VI and IX. However, workflow based systems so far fail in desiderata II, III, and IV.

Outside of workflow-based systems, very few database provenance systems have been applied to real-world scientific problems. However, we would like to highlight several systems that satisfy various desiderata. First, Trio [19] fulfills the notions for desiderata II, V and X very well. In [3], problems with desiderata III, V and X are explored. [20] are working on desideratum IV. Also, [7, 8] take an interesting look at desiderata II and V. Finally, [11] and [10] are both tackling desideratum IX.

4 Finding Practical Solutions

Each desideratum discussed above is a challenge to satisfy. In this section, we suggest how two of these challenges can be met.

4.1 Capturing Non-automated Processes

Human curators are often responsible for the content of specialized databases, or for "tweaks" in existing automated systems. The Uniprot consortium employs more than seventy scientists for curation. In some curated databases, the database designer augments the schema with provenance fields for the curator to populate; in "tweaked" systems, the actions often go unrecorded.

Using an appropriate architecture, and a language to express user actions, it is possible to capture these non-automated processes. By forcing a user to manipulate the database through a program that can track his movements, the user's actions can be distilled into Copy, Insert and Delete. Once we know (1) what action is occurring, (2) where in the current database the action is occurring, and (3) where any incoming data is from, we can effectively store provenance on the user's edits. Figures 2(a)–2(b) show an example series of user edits and their record in the provenance store. For further details, please refer to [6].

4.2 Reducing the Provenance Store

As stated in Desideratum VI, provenance information can balloon to gargantuan sizes. Utilizing features of the provenance store, it is possible to perform some reduction. A family of Factorization algorithms and two distinct Inheritance algorithms can reduce the provenance store by up to a factor of 20. Using Factorization, by breaking provenance records down into smaller pieces, it is possible to decrease repeated information. Using Inheritance, properties of the dataset are used to reduce the storage needed for the provenance. Figure 3 shows the ability of these algorithms to compress the provenance space needed for MiMI. Details of the algorithms and experiments can be found in [9].

5 Conclusions

The benefits of maintaining provenance are already apparent. Based on our experience with MiMI, we outline several desiderata that the next generation of provenance systems should meet. We outline some of the challenges in meeting these desiderata and suggest some directions to find a solution.

References

[1] J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster. Applying chimera virtual data concepts to cluster finding in the Sloan Sky Survey. *IEEE*, 2002.

[2] Roger S. Barga and Luciano A. Digiampietri. Automatic capture and efficient storage of escience experiment provenance. In *Concurrency and Computation: Practice and Experience*, 2007.

[3] Deepavali Bhagwat et al. An annotation management system for relational databases. In *Proc. of the Intl. Conf. on Very Large Data Bases (VLDB)*, pages 900–911, 2004.

[4] R. Bose and J. Frew. Composing lineage metadata with XML for custom satellite-derived data products. In *SSDBM*, pages 275–284, 2004.

[5] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, 2005.

[6] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *ACM SIGMOD*, pages 539–550, June 2006.

[7] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. In *ICDT*, pages 209–223, 2007.

[8] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[9] Adriane Chapman, H.V. Jagadish, and Prakash Ramanan. Efficient provenance storage. in submission, 2008.

[10] Kwok Cheung and Jane Hunter. Provenance Explorer - customized provenance views using semantic inferencing. In *International Semantic Web Conference*, pages 215–227, 2006.

[11] Shirley Cohen, Sarah Cohen Boulakia, and Susan Davidson. Towards a model of scientific workflows and user views. In *DILS*, pages 264–279, 2006.

[12] Ian Foster, Jens Vockler, Michael Eilde, and Yong Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *International Conference on Scientific and Statistical Database Management*, pages 37–46, July 2002.

[13] Ian Foster, Jens Vockler, M Wilde, and Yong Zhao. The virtual data grid: a new model and architecture for data-intensive collaboration. In *CIDR*, 2003.

[14] Juliana Freire, Claudio T. Silva, et al. Managing rapidly-evolving scientific workflows. In *IPAW*, 2006.

[15] Paul Groth, Simon Miles, and Luc Moreau. Preserv: Provenance recording for services. In *Proceedings of the UK OST e-Science second All Hands Meeting 2005 (AHM'05)*, 2005.

[16] Magesh Jayapandian, Adriane Chapman, et al. Michigan Molecular Interactions (MiMI): Putting the jigsaw puzzle together. *Nucleic Acids Research*, pages D566–D571, Jan 2007.

[17] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of recording and using provenance in e-science experiments. *Journal of Grid Computing*, 5(1):1–25, 2007.

[18] Luc Moreau, Bertram Ludäscher, et al. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 2007.

[19] Michi Mutsuzaki, Martin Theobald, et al. Trio-One: Layering uncertainty and lineage on a conventional DBMS. In *CIDR*, pages 269–274, 2007.

[20] M Seltzer, J Ledlie, C Ng, D Holland, K Muniswamy-Reddy, and U Braun. Provenance aware sensor data storage. In *NetDB*, 2005.

[21] Yogesh Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[22] Weijian Xuan, Pinglang Wang, Stanley J. Watson, and Fan Meng. Medline search engine for finding genetic markers with biological significance. *Bioinformatics*, 23:2477–2484, 2007.

Provenance in Scientific Workflow Systems

Susan Davidson,
Sarah Cohen-Boulakia, Anat Eyal
University of Pennsylvania
{susan, sarahcb, anate }@cis.upenn.edu

Bertram Ludäscher, Timothy McPhillips
Shawn Bowers, Manish Kumar Anand
University of California, Davis
{ludaesch, tmcphillips, sbowers, maanand}@ucdavis.edu

Juliana Freire
University of Utah
juliana@cs.utah.edu

Abstract

The automated tracking and storage of provenance information promises to be a major advantage of scientific workflow systems. We discuss issues related to data and workflow provenance, and present techniques for focusing user attention on meaningful provenance through "user views," for managing the provenance of nested scientific data, and for using information about the evolution of a workflow specification to understand the difference in the provenance of similar data products.

1 Introduction

Scientific workflow management systems (*e.g.*, myGrid/Taverna [18], Kepler [6], VisTrails [13], and Chimera [12]) have become increasingly popular as a way of specifying and executing data-intensive analyses. In such systems, a workflow can be graphically designed by chaining together tasks (*e.g.*, for aligning biological sequences or building phylogenetic trees), where each task may take input data from previous tasks, parameter settings, and data coming from external data sources. In general, a workflow specification can be thought of as a graph, where nodes represent *modules* of an analysis and edges capture the *flow of data* between these modules.

For example, consider the workflow specification in Fig. 1(a), which describes a common analysis in molecular biology: *Inference of phylogenetic (i.e., evolutionary) relationships between biological sequences*. This workflow first accepts a set of sequences selected by the user from a database (such as GenBank), and supplies the data to module M1. M1 performs a multiple alignment of the sequences, and M2 refines this alignment. The product of M2 is then used to search for the most parsimonious phylogenetic tree relating the aligned sequences. M3, M4, and M5 comprise a loop sampling the search space: M3 provides a random number seed to M4, which uses the seed together with the refined alignment from M2 to create a set of phylogenetic trees. M5 determines if the search space has been adequately sampled. Finally, M6 computes the consensus of the trees output from the loop. The dotted boxes M7, M8 and M9 represent the fact that *composite modules* may be used to create the workflow. That is, M7 is itself a workflow representing the alignment process, which consists of modules M1 and M2; M8 is a workflow representing the initial phylogenetic tree construction process, which consists of modules M3, M4, and M5; and M9 is a composite module representing the entire process of creating the consensus tree, which consists of modules M3, M4, M5 and M6.

The result of executing a scientific workflow is called a *run*. As a workflow executes, data flows between module *invocations* (or *steps*). For example, a run of the phylogenetics workflow is shown in Fig. 1(b). Nodes

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

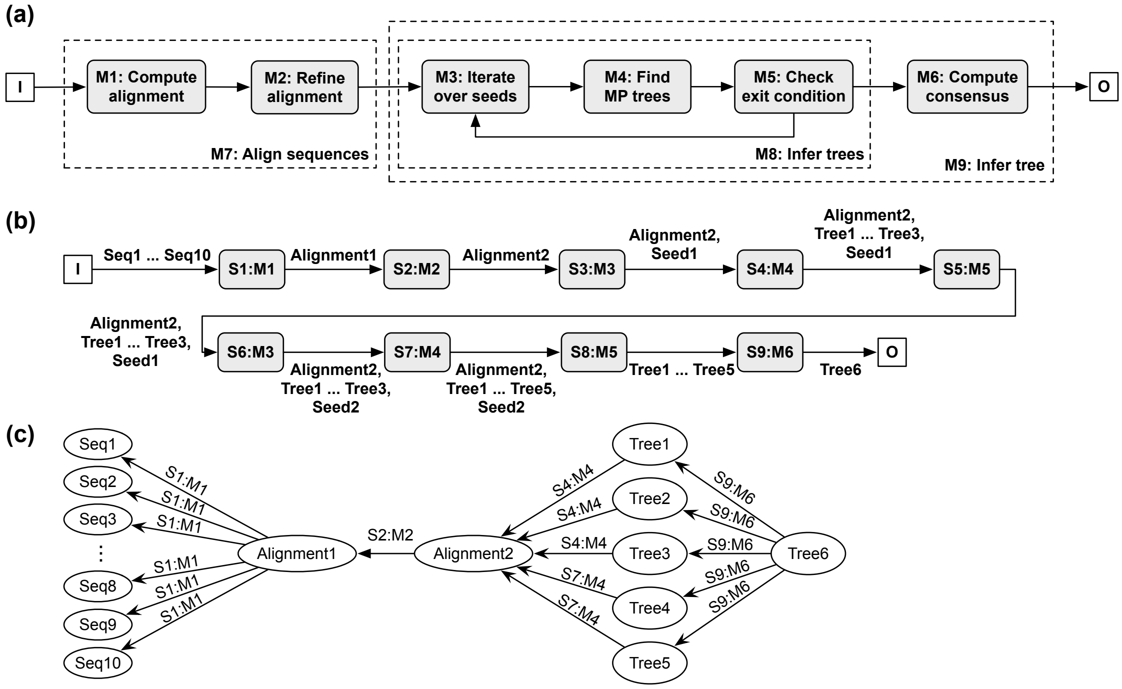


Figure 1: Phylogentics workflow specification, run, and data dependency graph.

in this run graph represent steps that are labeled by a unique step identifier and a corresponding module name (*e.g.*, S1:M1). Edges in this graph denote the flow of data between steps, and are labeled accordingly (*e.g.*, data objects `Seq1`,...,`Seq10` flow from input I to the first step S1). Note that loops in the workflow specification are always unrolled in the run graph, *e.g.*, two steps S4 and S7 of M4 are shown in the run of Fig. 1(b).

A given workflow may be executed multiple times in the context of a single project, generating a large amount of final and intermediate data products of interest to the user [9]. When such analyses are carried out by hand or automated using general-purpose scripting languages, the means by which results are produced are typically not recorded automatically, and often not even recorded manually. Managing such provenance information is a major challenge for scientists, and the lack of tools for capturing such information makes the results of data-intensive analyses difficult to interpret, to report accurately, and to reproduce reliably. Scientific workflow systems, however, are ideally positioned to record critical provenance information that can authoritatively document the lineage of analytical results. Thus, the ability to capture, query, and manage provenance information promises to be a major advantage of using scientific workflow systems. Provenance support in scientific workflows is consequently of paramount and increasing importance, and the growing interest in this topic is evidenced by recent workshops [4, 17] and surveys [5, 19] in this area.

Data provenance in workflows is captured as a set of dependencies between data objects. Fig. 1(c) graphically illustrates a subset of the dependencies between data objects for the workflow run shown in Fig. 1(b). In such data-dependency graphs, nodes denote data objects (*e.g.*, `Tree4`) and dependency edges are annotated with the step that produced the data. For example, the dependency edge from `Alignment2` to `Alignment1` is annotated with S2:M2 to indicate that `Alignment2` was produced from `Alignment1` as a result of this step.

Many scientific-workflow systems (*e.g.*, myGrid/Taverna) capture provenance information implicitly in an event log. For example, these logs record events related to the start and end of particular steps in the run and corresponding data read and write events. Using the (logical) order of events, dependencies between data objects processed or created during the run can be inferred[1]. Thus, determining data dependencies in scientific

[1]The complexity of the inference procedure and type of log events required depends on the specific model of computation used to

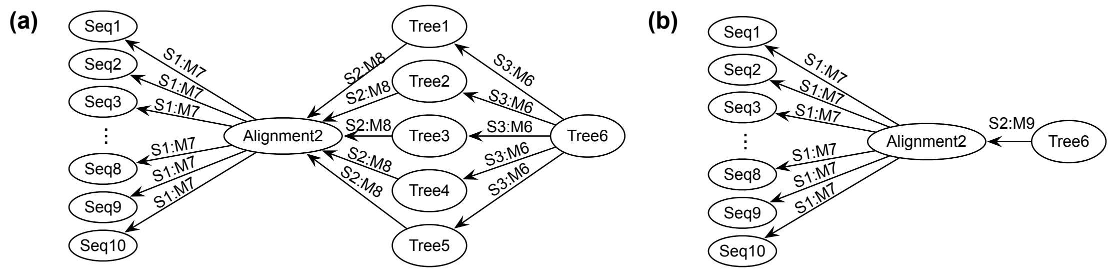


Figure 2: Provenance of `Tree6` in Joe's (a) and Mary's (b) user views.

workflow systems generally is performed using dynamic analysis, *i.e.*, modules are treated as "black boxes" and dependency information is captured as a workflow executes. In contrast, determining provenance information from database views (or queries) can be performed using static analysis techniques [10]. In this case, database queries can be viewed as "white box" modules consisting of algebraic operators (*e.g.*, σ, π, $\bowtie$). An intermediate type of provenance can also be considered in which black-box modules are given additional annotations specifying input and output constraints, thus making them "grey boxes" [6]. These additional specifications could then be used to reconstruct provenance dependencies using static analysis techniques, without requiring runtime provenance recording.

The use of provenance in workflow systems also differs from that in database systems. Provenance is not only used for interpreting data and providing reproducible results, but also for troubleshooting and optimizing efficiency. Furthermore, the application of a scientific workflow specification to a particular data set may involve tweaking parameter settings for the modules, and running the workflow many times during this tuning process. Thus, for efficiency, it is important to be able to revisit a "checkpoint" in a run, and re-execute the run from that point with new parameter settings, re-using intermediate data results unaffected by the new settings. The same information captured to infer data dependencies for a run can also be used to reset the state of a workflow system to a checkpoint in the past or to optimize the execution of a modified version of a workflow in the future.

While the case for provenance management in scientific workflow systems can easily be made, real-world development and application of such support is challenging. Below we describe how we are addressing three provenance-related challenges: First, we discuss how composite modules can be constructed to provide provenance "views" relevant to a user [3]. Second, we discuss how provenance for complex data (*i.e.*, nested data collections) can be captured efficiently [7]. Third, we discuss how the evolution of workflow specifications can be captured and reasoned about together with data provenance [13].

2 Simplifying provenance information

Because a workflow run may comprise many steps and intermediate data objects, the amount of information provided in response to a provenance query can be overwhelming. Even for the simple example of Fig. 1, the provenance for the final data object `Tree6` is extensive.[2] A user may therefore wish to indicate which modules in the workflow specification are *relevant*, and have provenance information presented with respect to that user view. To do this, composite modules are used as an abstraction mechanism [3].

For example, user Joe might indicate that the M2: *Refine alignment*, M4: *Find MP trees*, and M6: *Compute consensus* modules are relevant to him. In this case, composite modules M7 and M8 would automatically be constructed as shown in Fig. 1(a) (indicated by dotted lines), and Joe's user view would be $\{$M7, M8, M6$\}$. When answering provenance queries with respect to a user view, only data passed between modules in the user

execute a workflow, *e.g.*, see [8].

[2] The graph shown in Fig. 1(c) is only partial, and omits the seeds used in M4 as well as additional notations of S7:M4 on the edges from `Tree1`,..., `Tree3` to `Alignment2`.

view would be visible; data internal to a composite module in the view would be hidden. The provenance for `Tree6` presented according to Joe's user view is shown in Fig. 2(a). Note that `Alignment1` is no longer visible.

More formally, a *user view* is a partition of the workflow modules [3]. It induces a "higher level" workflow in which nodes represent composite modules in the partition (*e.g.*, M7 and M8) and edges are induced by dataflow between modules in different composite modules (*e.g.*, an edge between M7 and M8 is induced by the edge from M2 to M3 in the original workflow). Provenance information is then seen by a user with respect to the flow of data between modules in his view. In the Zoom*UserViews system [2], views are constructed automatically given input on what modules the user finds relevant such that (1) a composite module contains at most one relevant (atomic) module, thus assuming the "meaning" of that module; (2) no data dependencies (either direct or indirect) are introduced or removed between relevant modules; and (3) the view is minimal. In this way, the meaning of the original workflow specification is preserved, and only relevant provenance information is provided to the user.

Note that user views may differ: Another user, Mary, may only be interested in the modules M2: *Refine alignment* and M6: *Compute consensus*. Mary's user view would therefore be constructed as {M7, M9}, and her view for the provenance of `Tree6` (shown in Fig. 2(b)) would not expose `Tree1` ... `Tree5`.

3 Representing provenance for nested data collections

Modules within scientific workflows frequently operate over collections of data to produce new collections of results. When carried out one after the other, these operations can yield increasingly nested data collections, where different modules potentially operate over different nesting levels. The *collection-oriented modeling and design* (COMAD) framework [16] in Kepler models this by permitting data to be grouped explicitly into nested collections similar to the tree structure of XML documents. These trees of data are input, manipulated, and output by collection-aware modules. However, unlike a general XML transformer, a COMAD module generally preserves the structure and content of input data, accessing particular collections and data items of relevance to it, and adding newly computed data and new collections to the data structure it received. COMAD workflow designers declare the *read scope* and *write scope* for each module while composing the workflow specification. A read scope specifies the type of data and collections relevant to a module using an XPath-like expression to match one or more nodes on each invocation; paths may be partially specified using wildcards and predicates. As an example, the read scope for M1 could be given as `Proj/Trial/Seqs`, which would invoke M1 over each collection of sequences in turn. A write scope specifies where a module should add new data and collections to the stream. Data and collections that fall outside a module's read scope are automatically forwarded by the system to succeeding modules, enabling an "assembly-line" style of data processing.

Similar to other dataflow process networks [15], modules in a COMAD workflow work *concurrently* over items in the data stream. That is, rather than supplying the entire tree to each module in turn, COMAD streams the data through modules as a sequence of tokens. Fig. 3 illustrates the state of a COMAD run of the example workflow shown in Fig. 1 at a particular point in time, and contrasts the logical organization of the data flowing through the workflow in Fig. 3(a) with its tokenized realization at the same point in time in Fig. 3(b). This figure further illustrates the pipelining capabilities of COMAD by including two independent sets of sequences in a single run. This pipeline concurrency is achieved in part by representing nested data collections at runtime as "flat" token streams containing paired opening and closing delimeters to denote collection membership.

Fig. 3 also illustrates how data provenance is captured and represented at runtime. As COMAD modules insert new data and collections into the data stream, they also insert metadata tokens containing explicit data-dependency information. For example, the fact that `Alignment2` was computed from `Alignment1` is stored in the insertion-event metadata token immediately preceding the A2 data token in Fig. 3(b), and displayed as the dashed arrow from A2 to A1 in Fig. 3(a). The products of a COMAD workflow may be saved as an

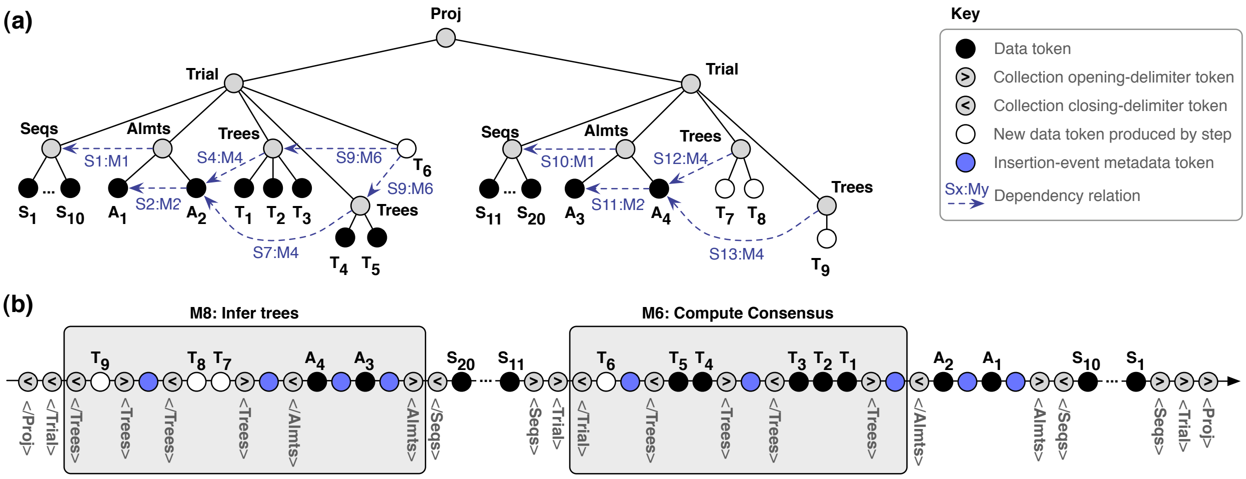


Figure 3: An intermediate state of a COMAD run

XML-formatted trace file, in which provenance records are embedded directly within the file as XML elements annotating data and collection elements. Detailed data dependencies can be inferred from the trace file, *i.e.*, from the embedded provenance annotations together with the nested data collections output by the workflow run. Note that COMAD can minimize the number and size of provenance annotations as described in [7, 9]. For example, when a module inserts a node that is a collection, the provenance information for that node implicitly cascades to all descendant nodes. Similarly, if a node is derived from a collection node, an insertion annotation is created that refers just to the collection identifier rather than the various subnodes.

The current COMAD implementation includes a prototype subsystem for querying traces. The system provides basic operations for accessing trace nodes, constructing dependency relations, and querying corresponding dependency graphs over the XML trace files. Methods also are provided to reconstruct parameter settings and metadata annotations attributed to data and collection nodes [7].

4 Workflow evolution

Scientific workflows dealing with data exploration and visualization are frequently exploratory in nature, and entail the investigation of parameter spaces and alternative techniques. A large number of related workflows are therefore created in a sequence of iterative refinements of the initial specification, as a user formulates and tests hypotheses. VisTrails [13] captures detailed information about this refinement process: As a user modifies a workflow, it transparently captures the change actions, e.g., the addition or deletion of a module, the modification of a parameter, the addition of a connection between modules,[3] akin to a database transaction log. The history of change actions between workflow refinements is referred to as a visual trail, or a *vistrail*.

The change-based representation of workflow evolution is concise and uses substantially less space than the alternative of storing multiple versions of a workflow. The model is also extensible. The underlying algebra of actions can be customized to support change actions at different granularities (e.g. composite modules versus atomic modules). In addition, it enables construction of an intuitive interface in which the evolution of a workflow is presented as a tree, allowing scientists to return to a previous version in an intuitive way, to undo bad changes, and be reminded of the actions that led to a particular result.

Vistrails and data provenance interact in a subtle but important way: The vistrail can be used to explain the difference in process between the data provenance of similar data products. Returning to our example,

[3]Modules are connected by input/output *ports*, which carry the data type and meaning. Static type-checking can be therefore performed to help in debugging.

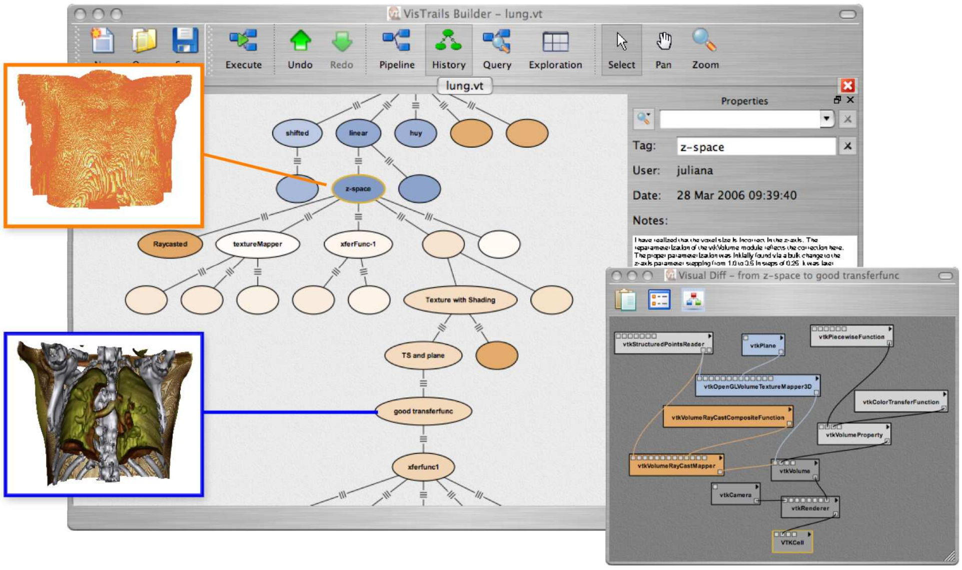


Figure 4: Visual difference interface for a radiation treatment planning example.

suppose that two runs of the workflow in Fig. 1 took as input the same set of sequences, but returned two different final trees. Furthermore, suppose that the specification was modifed between the two runs, e.g. that a different alignment algorithm was used in M1, or that three iterations of the loop were performed in M8 due to different seeds being used. Rather than merely examining the data provenance of each tree, the scientist may wish to compare their provenance and better understand *why* the final data differed. However, computing the differences between two workflows by considering their underlying graph structure is impractical; the related decision problem of *subgraph isomorphism (or matching)* is known to be NP-complete [14]. By capturing evolution explicitly in a vistrail, discovering the difference in process is simplified: The two workflow nodes are connected by a path in the vistrail, allowing the difference between two workflows to be efficiently calculated by comparing the sequences of change actions associated with them [11].

Figure 4 (right) shows the visual difference interface provided by VisTrails. A visual difference is enacted by dragging one node in the history tree onto another, which opens a new window with a difference workflow. Modules unique to the first node are shown in orange, modules unique to the second node in blue, modules that are the same in dark gray, and modules that have different parameter values in light gray. Using this interface, users can correlate differences between two data products with differences between their corresponding specifications.

5 Conclusion

Workflow systems are beginning to implement a "depends-on" model of provenance, either by storing the information explicitly in a database (*e.g.*, VisTrails) or within the data itself (*e.g.*, COMAD). Several techniques have also been proposed to reduce the amount of provenance information either presented to the user (*e.g.*, user views), or stored by the database (*e.g.*, by treating data as collections). Furthermore, since workflow specifications evolve over time, there is a need to understand not only the provenance of a single data item but how the provenance of related data items differ.

Although some workflow systems provide a query interface for interacting with the provenance information, it is still an open problem as to what a provenance query language should provide. For example, we might wish to scope provenance information within a certain specified portion of a workflow, or return all provenance information that satisfies a certain execution pattern. The query language should also allow users to issue high level queries using concepts that are familiar to them, and present the results in an intuitive manner. Related work in this area has been done in the context of business processing systems, in which runs are monitored by querying logs (*e.g.*, [1]).

References

[1] C. Beeri, A. Pilberg, T. Milo, and A. Eyal. Monitoring business processes with queries. In *VLDB*, 2007.

[2] O. Biton, S. Cohen-Boulakia, and S. Davidson. Zoom*UserViews: Querying relevant provenance in workflow systems (demo). In *VLDB*, 2007.

[3] O. Biton, S. Cohen-Boulakia, S. Davidson, and C. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, 2008 (to appear).

[4] R. Bose, I. Foster, and L. Moreau. Report on the International Provenance and Annotation Workshop. *SIGMOD Rec.*, 35(3), 2006.

[5] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comp. Surveys*, 37(1):1–28, 2005.

[6] S. Bowers and B. Ludäscher. Actor-oriented design of scientific workflows. In *ER*, 2005.

[7] S. Bowers, T. M. McPhillips, and B. Ludäscher. Provenance in collection-oriented scientific workflows. In *Concurrency and Computation: Practice and Experience*. Wiley, 2007 (in press).

[8] S. Bowers, T. M. McPhillips, B. Ludäscher, S. Cohen, and S. B. Davidson. A model for user-oriented data provenance in pipelined scientific workflows. In *IPAW*, volume 4145 of *LNCS*, pages 133–147. Springer, 2006.

[9] S. Bowers, T. M. McPhillips, M. Wu, and B. Ludäscher. Project histories: Managing provenance across collection-oriented scientific workflow runs. In *Data Integration in the Life Sciences*, 2007.

[10] P. Buneman and W.Tan. Provenance in databases. In *SIGMOD*, pages 1171 – 1173, 2007.

[11] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. Using provenance to streamline data exploration through visualization. Technical Report UUSCI-2006-016, SCI Institute–University of Utah, 2006.

[12] I. Foster, J. Vockler, M. Woilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *SSDBM*, pages 37–46, 2002.

[13] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *IPAW*, 2006.

[14] J. Hastad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[15] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[16] T. M. McPhillips, S. Bowers, and B. Ludäscher. Collection-oriented scientific workflows for integrating and analyzing biological data. In *Data Integration in the Life Sciences*, 2006.

[17] L. Moreau and B. Ludäscher, editors. *Concurrency and Computation: Practice and Experience – Special Issue on the First Provenance Challenge*. Wiley, 2007 (in press). (see also http://twiki.ipaw.info/bin/view/Challenge/).

[18] T. Oinn *et al.* Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(1), 2003.

[19] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, 2005.

Copyright and Provenance: Some Practical Problems

John Mark Ockerbloom
University of Pennsylvania
ockerblo@pobox.upenn.edu

Abstract

Copyright clearance is an increasingly complex and expensive impediment to the digitization and reuse of information. Clearing copyright issues in a reliable and cost-effective manner for works created in the last 100 years can involve establishing complex provenance chains for the works, their copyrights, and their licenses. This paper gives an overview of some of the practical provenance-related issues and challenges in clearing copyrights at large scale, and discusses efforts to more efficiently gather and share information and its copyright provenance.

1 Introduction

As information seekers increasingly move from print to digital media, print resources are being digitized at an ever-accelerating rate. As of 2007, over a million volumes have been digitized by libraries such as the Library of Congress and the University of Michigan, for-profit corporations like Google, and public-private partnerships such as the Open Content Alliance [1]. Mass digitization is made possible by ever-lower costs for large-scale scanning and storage. The Open Content Alliance's scanning projects for example, digitize books nondestructively at a cost of 10 cents per page, or about $30 for a 300-page book [2].

The cost of clearing copyright, however, can be substantially higher than the cost of digitization itself. A 2003 study of attempts to obtain copyright permissions for a book digitization project at Carnegie Mellon University found that it cost $78 per title to clear copyrights of the copyrighted books they sought to digitize [3]. This figure does not include any royalty costs, but only the overhead cost in determining copyright status and obtaining necessary permissions. Most of this cost was labor, a cost that tends to increase over time.

Most books, particularly those by a single author, have relatively few and simple copyrights. However, periodicals, collective works, sound recordings, and motion pictures often involve a large number of potential copyrights and copyright owners in their various elements. Moreover, different rights and permissions may apply to these copyrights in different contexts and legal jurisdictions.

There is widespread scientific, business, and cultural interest in disseminating, adapting, and reusing the content of others, as seen in initiatives like the Internet Archive, Google Book Search, YouTube, and Arxiv.org. Since copyright restrictions apply to most present-day content, as well as historic content going as far back as 100 years, clearing copyright can be both an essential and a costly part of these initiatives.

In order to legally publish and reuse content, one typically needs to determine the answers to several important questions:

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- What copyrights, if any, apply to this content? (And which are currently in force?)
- Who controls these copyrights, and how can they be contacted?
- What permissions, if any, have been granted concerning the copyrighted content?

In projects that make large-scale reuse of content, it may not be feasible to have complete, certain answers to these questions for all the content one may want to use. In practice, content reuse at scale may be best understood as an optimization problem along multiple dimensions, according to these desires:

- **Maximizing the value of the collection**, by including as many valuable resources as one can, with the broadest rights possible.
- **Maximizing throughput on rights-clearing**, so a large collection can be built quickly.
- **Minimizing the cost of rights-clearing**, which as noted above can can be impractically high per item.
- **Minimizing the risk of legal penalties**, which in the worst case can be very large. Current copyright law in the US authorizes statutory penalties (which are distinct from penalties for actual damages) of up to $150,000 per infringement. Penalties are lower if the infringement is shown not to be willful, but proving that in court can be uncertain and costly, and even non-willful infrigement statutory penalties can run into the thousands of dollars [4].

Different projects may put different priorities on these dimensions. For example, Google's Library project has to date been especially conservative with copyright determination in its "full view" book displays. In some cases it presumes that US copyright is still in force for books published as long ago as 1909, and that foreign copyrights may subsist from as far back as 1865. (In a large number of these cases, these copyrights have in fact long since expired.) These conservative guidelines, intended to minimize clearance cost and risk, enable them to make visible a significant number of books with very little effort needed to clear titles, but suppresses much content that could be usefully viewed and repurposed by the public. In contrast, organizations like the Internet Archive put a higher priority on maximizing collection value in the copyright clearance of their digitized texts, and therefore expose many commercially published works from as late as 1963, and government-published works up to the present day.

In order to control risk and cost of clearing copyright in a large-scale project, simply stating copyright restrictions in binary terms (such as "this is in the public domain" or "this is in copyright" is insufficient. To understand the reliability and applicability of such determinations, one needs to know a variety of facts that inform those determinations, and know how these facts were derived. Moreover, the same facts may lead to different determinations in different places, times, and contexts. What may be legal to reuse in a classroom in the US in 2007 may not be legal to reuse in a commercial film in Japan in 2009, or vice versa, but the determination in both of these contexts may depend on the same underlying set of facts about the work in question and its copyrights.

Managing these facts involves several kinds of provenance issues. To reliably determine the rights to a work, one may have to understand and record the provenance of a work, the provenance of its rights, and the provenance of the information used in rights determination. In the next sections, I survey some of the specific provenance problems involved, and describe some of the derivations and uncertainties that are part of copyright clearance. I then describe some methods to alleviate the problems of provenance in copyright determinations, and suggest ways in which copyright clearance can be a productive and illuminating application domain for provenance research.

2 Provenance issues

In this section, I describe some of the provenance-related issues involved in determining a work's copyright status and the rights available for using the work. It is not within the scope of this paper to provide a complete or authoritative guide to copyright clearance; rather, I illustrate important aspects of clearance where provenance is relevant. Useful detailed guides to copyright clearance in the US, written by copyright attorneys, include [5] and [6].

2.1 Provenance of works

Whether a work is copyrighted at all, what copyrights might apply to it, and who initially owns the copyrights to the work, depend on the provenance of the work itself. Relevant questions include:

- **Who authored the work?** And when did they live? Copyrights are generally assigned to authors by default, and in many cases last for a set period after the author's death.
- **When was the work created, first published, first published with a US-recognized copyright notice, and first published in the US?** In various cases, the time and place of these events determine when a copyright term starts and ends.
- **Is this a work for hire? For whom?** Works for hire may have different initial copyright owners and copyright term lengths. Works produced as work for hire for the US government may not be copyrighted at all.
- **Does this work include or derive from other works?** If so, the rights available for this work may depend on the rights available for those other works.
- **What is the work's current commercial status?** For example, if a book is sufficiently old, out of print, and cannot be bought inexpensively, some US users may have special rights to reuse the work without permission under special provisions of US copyright law, even if the work is still under copyright [7].

2.2 Provenance of rights

The rights available for the use of a work depend on a number of factors apart from the provenance of the work itself. The provenance of the rights must also be considered. In some jurisdictions, for instance, copyrights must be explicitly asserted and maintained through various mechanisms in order to remain in force. Rights to a work can also be transferred, in whole or in part, from the original author to new agents. (This is commonly done with articles submitted to scholarly journals, for example.) The original or subsequent rightsholders can further license the work under various terms and conditions. These transfers and licenses may be matters of public record, or private agreement.

Under the Berne Convention, the dominant international copyright treaty, copyright automatically applies to a new creation without any formal claims or registration. While this principle may make it easier to determine that a copyright has not prematurely expired, it may make it more difficult to determine when the copyright was originally established, and who claimed it.

Many copyrights today, however, were originally established under different regimes than that of Berne. For example, the United States was a latecomer to the Berne treaty and until the early 1990s made copyright status dependent on notices, registration, and formal renewal of copyright. Under US law, valid copyright notices include an explicit claim of copyright, a year in which the copyright was claimed, and the name of the claimant. Registration and renewal involves providing certain data about the work, including the author, title, and date of claim. The US Copyright Office records, maintains and makes available the data in these registrations.

Relevant questions of rights provenance, then, include:

- **What copyright notices, if any, were distributed with the work?** These notices may determine the copyright status of a work, as well as noting the initial owner and the start of the work's copyright term.
- **What registrations and renewals were made of the copyright?** These may also determine the copyright status of a work and identify owners. Renewals may show ownership changes since the initial registration.
- **What assignments were made of the copyright or of subsidiary rights?** These may involve full copyright transfers, or narrower assignment of rights for certain uses and jurisdictions. For example, a freelance writer may assign US first serial rights to an article to a particular magazine, but retain rights to republish the work in other formats and markets. The exact terms of such rights assignments are typically governed by contract language rather than by statute, though in some cases, such as intestate or insolvent authors, rights assignments may be determined by local inheritance or bankruptcy laws. Rights may be assigned to a specific party, or in some cases to the world at large. Open source and Creative Commons licenses, for example, specify that anyone has certain rights to use a work under standard terms and conditions that are typically published along with the work.

2.3 Provenance of information

WhIle provenance of the work and the rights are sufficient in theory to determine whether and how a work can be used, in practice one cannot rely on perfect and complete knowledge of this information. Therefore, those who wish to make copyright determinations must also consider the provenance of the information they have about the works and copyrights. What are the sources of information? Are they reliable? Did they derive their information from other sources? If so, which ones? Are there important sources of information that are not being taken into account, and could these change one's copyright determinations in important ways?

Many rights determination issues include or derive from negative as well as positive information. For example, consider the judgment that a book first published in 1940 in the US is in the public domain in that country. Positive information supporting this judgment may include the imprint of a US publisher, and the notations "first edition" and "copyright 1940" on the title and verso pages of the book. Negative information may include the lack of any prior editions of the book, the lack of any further copyright notices in the book, the lack of a copyright renewal, and the lack of any prior publications from which the book derives.

Some of these facts may be easier to establish than others, with negative information usually more difficult to prove than positive information. The imprint and copyright notice of a book, for instance, can be verified with images of the pages on which they appear. The lack of other copyright notices might be established from other pages on which copyright notices might appear, which can be a larger page set. The lack of copyright renewal can be verified against a complete data source of copyright renewals, which exists, but which is in turn a much larger information set that is more difficult to access or search in full than the information sets discussed to this point. The lack of previous editions or works from which the book might derive depends on yet larger, and less well-defined, information domains. In practice, at some point along the continuum of verifying these facts, one will need to rely on the judgment of another person or source, rather than including the complete set of information needed to establish a particular fact.

The particular source of this information is important. One might trust the word of a publisher or a professional librarian about the copyright date or status of a book more than the word of an anonymous uploader to a file-sharing site. However, sometimes unexpected information can surprise even experts. In 2004, a popular online animated political satire used, without permission, the tune and some of the words of Woody Guthrie's song "This Land is Your Land" The animators were sent a cease and desist notice by Guthrie's music publishers, who had duly registered and renewed the copyright on their first edition of the work. A complaint brought by the Electronic Frontier Foundation on behalf of the producers of the animation initially relied on a fair use defense; however, in the course of litigation, the Foundation discovered that Guthrie had produced a hand-written song book, complete with copyright notice and cover price, that included an early version of the song, years before the song was conventionally published. Only a few known copies of this work are known to exist, and its copyright was never renewed. The publisher and the animators settled quickly thereafter, with an agreement allowing the online animation to continue [8].

In some cases, positive or negative information asserted about the copyright of a work may need to be discounted or overridden. It is not unheard of, for example, for a publisher to place a copyright notice, dated the year of publication, on an unaltered reprint of a public domain work, even though under US law such reprints are not entitled to a new copyright. Contributors to shared content sites like YouTube or Wikipedia may attach liberal licenses to content, authored by others, that they do not have the right to relicense or redistribute. For some uses, such as the literal reproduction of works that carry copyright notices, it may be important to retain these assertions while at the same time noting that they do not apply, or do not apply in full.

3 Derivations and uncertainties

How far back one needs to trace and record the provenance of copyright information depends on the relative importance of risk, cost, and productivity in the optimization problem described earlier. Their importance, and the degree of copyright provenance recording required, may vary depending on context and application. Project Gutenberg, for instance, requires and stores title and verso page images to establish original claims of copyright. It also has produced, and uses, a text transcription of the book renewal sections of the US Copyright Office's Catalog of Copyright Entries, in order to determine whether a book copyright has been renewed. For other fact determinations relevant to copyright, however, such as the lack of prior publications of the work, it relies on the judgment and assertions of its contributors and volunteers.

The transcription of the Catalog of Copyright Entries itself is derived from earlier artifacts. The ultimate source of a copyright renewal claim is the renewal form filed by a copyright holder and deposited with the US Copyright Office. These forms are included or reproduced in registration books that are accessible to Copyright Office staff. From 1978 onward, the information in these forms has been used to populate an online database accessible worldwide. Before 1978, though, catalog cards were prepared from these forms to allow copyright registrations and renewals to be looked up by name, title, or various other criteria. These cards are accessible to both the Copyright Office staff and to members of the general public that can visit the copyright card catalog in Washington, DC. From these cards, bound volumes of the Catalog of Copyright Entries were printed and distributed to libraries across the United States and beyond, where they are available to patrons of those libraries (though in many libraries the volumes are kept in closed reserve stacks). From this point, already some distance down the provenance chain, independent third parties have made digital images of some of the Catalog of Copyright Entries pages and published them on the Internet, where the digital images have been used to produce text transcriptions of the copyright registration information that are used by Project Gutenberg and others.

The renewal records that can be quickly searched online, then, can be information derived via several steps from the original copyright holder filings. It is possible that errors or omissions exist in the derivations, and that erroneous rights determinations may be made as a result. Going back further brings one closer to the original copyright filings, but is generally more difficult and costly. As mentioned above, the Catalog of Copyright Entries have been partially digitized. The copyright card catalog has not, nor have the registration books. Digitizing these resources would be more expensive than digitizing the summary Catalog, due to the larger number of images and the rarity and vulnerability of the materials. They would also be more cumbersome to search than a database would be. Thus, we see that following provenance chains further back involves a tradeoff of cost and risk. Studies at Stanford suggest that the error rate of using transcriptions of the Catalog of Copyright Entries is very low, and that errors in derivation were much less common than errors and term mismatches in copyright searches [9]. Many projects, then, may well find the more easily searched derivative forms of the copyright records a useful or even superior starting point for research.

In many cases, simply searching copyright records will not answer the question of what can be done with a work. If the records indicate that a work is still under copyright, or there is not sufficient information to determine with sufficient certainty that a work is no longer under copyright, then one may be legally liable if one reuses the work. To avoid this liability, one must obtain permission (or assurances of public domain status) from the presumed rightsholder. Unfortunately, for many works the rightsholder can be difficult or impossible to determine or contact. Copyright claimants at the time of registration may be listed in the Catalog of Copyright Entries, but their addresses are not (though they may appear in the registration books). Copyrights may have been transferred, assigned or willed to others since the copyright was registered or renewed, and this is more likely the longer that the copyright term has run. While copyright transfers may be registered with the Copyright Office, there is no requirement that transfers be registered there or anywhere else.

Hence, there is now a large and growing set of "orphan works" for which copyright cannot be reliably cleared, due to the inability to determine or locate the proper copyright owners. Orphan works come in all varieties: "abandonware" developed by defunct software companies; articles and monographs from long-dead authors with obscure heirs; images of great historic or artistic importance whose original creator cannot be traced; documentary productions and compilations that feature copyrighted material from a wide variety of creators, not all of whom can be determined or found. As US law stands now, orphan works are effectively impossible to reuse legally (beyond the usual rights of fair use and resale). The Copyright Office has acknowledged the orphan works problem as a serious one, and has held public hearings and suggested legislation to alleviate it.

Uncertainties can also exist with rights to data. In many European countries, factual data can have copyright-like restrictions associated with it. (In the US, facts in themselves are in the public domain, though an original selection, expression, or arrangement of the facts can be copyrighted.) Also, in many legal jurisdictions, privacy laws or confidentiality agreements may limit the disclosure of certain data. Keeping track of rights and restrictions on private information, while often not specifically a copyright issue, involves many of the same issues of provenance tracking as copyright clearance does.

4 Initiatives for easing copyright clearance

Copyright clearance need not be as complicated and risky as it currently is. Several initiatives have been started or proposed to ease copyright clearance, a number of which relate to provenance.

One way to ease copyright clearance at the large scale is simply to promulgate standards for recording and distributing copyright-related information. For example, many of the digitized books at the Internet Archive include metadata relevant to copyright in standard forms, including publication dates, copyright notice information, and the results of copyright renewal searches. While this information is not currently formatted for machine processing, its use of a standard vocabulary and set of assertions about copyright has inspired efforts to define standard, structured, machine-readable vocabularies and grammars for expressing copyright facts [10]. Note that the vocabulary of copyright provenance is different from the vocabulary of digital restrictions often used by Digital Rights Management (DRM) systems. The latter is largely concerned with the specific operations that software should allow or disallow on particular content, such as printing, reading aloud, or duplicating. The former is concerned with underlying intellectual rights and permissions, such as the existence and owner of copyright, the duration of the copyright, and licenses granted for the content. DRM restrictions may be derived in part from these underlying rights, but are distinct from them.

One vocabulary of rights expression that has gained widespread popularity in recent years is the Creative Commons vocabulary. Creative Commons supports a variety of standardized permissions, such as the right to reuse with attribution, or noncommercially, or without making derivatives, or with the right to make derivatives that must be licensed under the same terms as the original. These permissions can be encoded in machine-readable format and distributed along with, or as part of, a copyrighted work. Assuming that the permissions were granted by an authorized party, this rights expression system allows others to easily reuse a work in well-understood ways, without having to trace the copyright holder to get special permissions [11].

Registries are useful as stores of copyright clearance data, including provenance data. The US Copyright Office is one such registry already discussed, but other types of registries also exist or have been proposed. For example, the Writers, Artists, and Their Copyright Holders registry, based in the US and the UK, keeps up-to-date contact information on many well-known copyright holders [12]. Groups representing copyright holders, such as the Copyright Clearance Center and the Harry Fox music licensing agency, both track copyright holders of works and handle payments to them, streamlining common uses of many copyrighted works such as recording new performances of songs or reprinting articles from periodicals. The Online Computer Library Center (OCLC) has proposed a general purpose registry of copyright information to be associated with its WorldCat union catalog, to make it easier to clear rights for all kinds of library materials [13].

Legislation can also ease copyright clearance, and suggest particular types of provenance information that may be useful to track. For example, the proposed Orphan Works Act of 2006 would have allowed copyrighted works to be reused by others without permission if the users were unable to find the copyright holder after a "reasonably diligent search" [14]. To date, orphan works legislation has not been enacted in the US, but if it were, it could limit the degree of provenance information one would need to gather for a work (since one might not have to trace back copyright information indefinitely if doing so were unreasonably burdensome). On the other hand, to prove that a reasonably diligent search was conducted, users might wish to explicitly document the steps taken in the search, to establish the provenance of one's "reasonably diligent" determination.

5 Conclusions

The preceding discussion should illustrate how provenance issues are important in copyright clearance, and how copyright clearance is a significant application domain for provenance research. Practical, reliable copyright clearance requires careful consideration of the provenance of works, their rights, and the assertions about the works and rights. Optimal procedures for rights determination must take into account positive and negative factual assertions, legal analysis tailored to jurisdiction and context, and an appropriate balancing of value, throughput, cost and risk. The factual assertion chains that support determinations of rights available for works can be complex in their structure, and involve varying degrees of uncertainty.

Copyright clearance, then, is fertile ground for applying provenance research. Theoretical foundations for evaluating the reliability of assertion chains can be applied to estimate risk in copyright determinations. Common representations of copyright assertions, searches, and their provenance, can be preserved as metadata and used in context-sensitive copyright evaluations. Simple, cheap methods of storing and querying this provenance information, and improvements in the efficiency of provenance calculations, data representations, and queries, can improve the reliability and practicality of copyright evaluations in large-scale collections.

Moreover, improved copyright clearance is not simply an interesting research application. The easier it is to safely and legally reuse the works of the past, the easier it becomes to advance the state of knowledge and culture. The technologies that now allow organizations to digitize millions of books for the Internet make it possible to revive, redistribute and build upon the large corpuses of text, data, audiovisual media, and software, that make up the historic, cultural, and scientific endowment of the world. If advances in provenance handling allow us to more easily clear their copyrights, we may all enjoy greater access to a richer heritage of knowledge. As Isaac Newton and other scientists have noted, building on this richer heritage can let us all see farther, standing on the shoulders of giants [15].

References

[1] Sharita Forrest, “An Open Book: CIC Member Libraries Join Google in Digitizing up to 10 Million Volumes” *Inside Illinois*, 26:22, June 2007, online at http://www.news.uiuc.edu/ii/07/0621/google.html.

[2] “Open Content Alliance Will Scan Boston’s Books” *Chronicle of Higher Education*, September 28, 2007; online at http://chronicle.com/wiredcampus/article/2418/open-content-alliance-will-scan-the-best-of-bostons-books .

[3] Denise Troll Covey, *Acquiring Copyright Permission to Digitize and Provide Open Access to Books.* Digital Library Federation and Council on Library and Information Resources, October 2005. Online at http://purl.oclc.org/dlf/pubs/dlf105/

[4] Statutory penalties are specified in section 504 of the Copyright Act, “Remedies for Infringment: Damages and Profits” (17 USC 504). Online at http://www.copyright.gov/title17/92chap5.html#504 .

[5] Stephen Fishman, *The Public Domain: How to Find Copyright-Free Writings, Music, Art and More.* (3rd edition; Berkeley, CA: Nolo Press, 2006.)

[6] William S. Strong, *The Copyright Book: A Practical Guide.* (5th edition; Cambridge, MA: MIT Press, 1999.)

[7] See section 108 of the Copyright Act, “Limitations on exclusive rights: Reproduction by libraries and archives” (17 USC 108), online at http://www.copyright.gov/title17/92chap1.html#108 . The right of libraries to reproduce older works no longer commercially exploited under certain conditions is spelled out in section 108(h).

[8] Katie Dean, “JibJab is Free for You and Me” Wired News, August 24, 2004; online at http://www.wired.com/entertainment/music/news/2004/08/64704 .

[9] Mimi Calter, “Assessing Copyright Status: Copyright Renewals Database” Presentation at Digital Library Federation Forum, November 2007. Online at http://www.diglib.org/forums/fall2007/presentations/Calter.pdf .

[10] “Data elements needed to ascertain copyright facts” MARC Discussion Paper No. 2007-DP05, May 30, 2007. Online at http://www.loc.gov/marc/marbi/2007/2007-dp05-original.html .

[11] Creative Commons website. Online at http://creativecommons.org/ .

[12] The Watch File: Writers, Artists and Their Copyright Holders, online at http://tyler.hrc.utexas.edu/ .

[13] See comment by Bill Carney of OCLC in O’Reilly Radar, November 7, 2007, online at http://radar.oreilly.com/archives/2007/11/checking_copyri.html .

[14] Gigi Sohn, “Orphan Works Bill Introduced” Public Knowledge Policy Blog, May 22, 2006. Online at http://www.publicknowledge.org/node/392 .

[15] Newton’s famous aphorism, used by numerous writers both before and after his time, is discussed in great detail in Robert K. Merton’s book *On the Shoulders of Giants: A Shandean Postscript* (Chicago: University of Chicago Press, 1993). I verified the aphorism, and Newton’s use of it, with a searchable digital copy of the book provided by Amazon at http://www.amazon.com/exec/obidos/ASIN/0226520862 .

ICDE'08

24th IEEE International Conference on Data Engineering

April 7-12, 2008, Cancún, México

CALL FOR PARTICIPATION

Data Engineering deals with the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments.

The **24th IEEE International Conference on Data Engineering** will provide a forum for:

- sharing research solutions to problems of today's information society
- exposing practitioners to the latest research, tools, and practices
- raising awareness in the research community of the problems and challenges of practical applications of data engineering
- promoting the exchange of data engineering technologies and experience among researchers and practitioners
- identifying new issues and directions for future research and development work
- the exchange and discussion of new ideas and for interacting/networking with peers

HIGHLIGHTS

- 3 ***Keynotes***
- 6 ***Advanced Technology Seminars***
- 75 ***Research*** papers with *full presentations*, and 44 ***Research*** papers with *short presentations*, out of over 600 submissions
- 10 ***Workshops*** in conjunction with the main conference
- 3 ***Panels***
- 75 ***Posters*** out of 650 submissions
- 13 ***Industrial*** papers, out of over 50 submissions
- 23 ***Demos,*** out of over 60 submissions

KEYNOTES

- Hector Garcia-Molina (Stanford University, USA), on ***PhotoSpread: A Spreadsheet for Managing Photos***
- Martin Kersten (CWI, The Netherlands), on ***The Database Architecture Jigsaw Puzzle***
- Luis von Ahn (Carnegie Mellon University, USA), on ***Human Computation***

WORKSHOPS

- ***Self-Managing Database Systems*** (SMDB)
- ***Mining Multimedia Streams in Large-Scale Distributed Environments*** (MMSDE)
- ***RFID Data Management*** (RFDM)
- ***Ranking in Databases*** (DBRank)
- ***Methodologies, Architectures and Systems for Information Integration*** (IIMAS)
- ***Secure Semantic Web*** (SSW)
- ***Data and Services Management in Mobile Environment***s (DS2ME)
- ***Networking Meets Databases*** (NetDB)
- ***Data Engineering for Blogs, Social Media, and Web 2.0***
- ***Similarity Search and Applications*** (SISAP)

ADVANCED TECHNOLOGY SEMINARS

- ***Mobile and Embedded Database Systems and Technology***

Anil Nori (Microsoft Corp.)

- ***Data and Metadata Alignment: Concepts and Techniques***

Lise Getoor (University of Maryland) and Renee Miller (University of Toronto)

- ***Exploring the Power of Links in Scalable Data Analysis***

Jawei Han (University of Illinois, Urbana-Champaign), Xiaoxin Yin (Google) and Philip Yu (IBM T.J. Watson)

- ***Stream Processing: Going Beyond Database Management Systems***

Sharma Chakravarthy (University of Texas, Arlington)

- ***The Java Persistence API (JPA): Technology, Standards, and Implementations***

Patrick Linskey (BEA Systems, Inc.)

- ***Performance Evaluation in Database Research: Principles and Experience***

Ioana Manolescu (INRIA Futurs) and Stefan Manegold (CWI)

Venue: CANCUN

ICDE 2008 will take place at **Cancún**, a truly unique spot nestled in the heart of the Mexican Caribbean. Cancún's Hotel Zone is a 14 mile long island shaped like a "7" and connected to the mainland by bridges at either end. Some 25% of this area's natural surroundings are protected in ecological reserves and holds the second largest reef system in the world. Part of its cultural heritage can be seen in dozens of remains of ancient Mayan cities—some more than 2,600 years old—encompassing more than 5,000 buildings or temple mounds.

The conference hotel, **Presidente InterContinental Cancún Resort**, is all you could ever want in a tropical retreat: five-star luxury, premium restaurants and the best, picture-perfect beach in the peninsula. Central shops and nightlife are 10 minutes away, Isla Mujeres is just across the water, and there's a golf course next door.

For more information, visit www.icde2008.org

IEEE COMPUTER SOCIETY · IEEE · hp invent · Microsoft · CIC · IBM · SAP · YAHOO!

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398