

Provenance, XML, and the Scientific Web

James Cheney

University of Edinburgh
jcheney@inf.ed.ac.uk

Abstract

Science is now being revolutionized by the capabilities of distributing computation and human effort over the World Wide Web. This revolution offers dramatic benefits but also poses serious risks due to the fluid nature of digital information. The Web today does not provide adequate repeatability, reliability, accountability and trust guarantees for scientific applications. One important part of this problem is tracking and managing provenance information, or metadata about sources, authorship, derivation, or other historical aspects of data. In this paper, we discuss motivating examples for provenance in science and computer security, introduce four models of provenance for XML queries, and discuss their properties and prospects for further research.

1. Introduction

Historically, scientific computation has focused on perform large-scale numerical computations and simulations as quickly and accurately as possible. These calculations are well-understood mathematically, and thus, boring in a certain sense.

Scientific computation is no longer boring. Systems that mix numerical and symbolic computation, such as databases, or that involve coordination of programs distributed over ad hoc networks of heterogeneous systems. Such systems have historically been employed primarily for commercial purposes but are now being used intensively in several branches of science. For example, biologists now use database management systems [23] to collect large volumes of bibliographic and experimental data in order to perform new kinds of longitudinal studies or to guide discovery. Similarly, astronomers and physicists are building “Grids” [25] that combine database systems and distributed computing resources to store and process large volumes of data obtained from detailed sky surveys or from new instruments such as the Large Hadron Collider, respectively.

These applications exemplify an emerging reality: computer systems are now used to represent symbolic knowledge, not just perform numerical scientific calculations. Various funding agencies have conceived terms for this phenomenon, such as *cyberinfrastructure* or *eScience*. However, these buzzwords arguably understate the true significance of this revolution: scientific knowledge (and conclusions) are now produced using radical sharing of content and distribution of effort over the Web.

This practice has dramatic advantages but also introduces serious risks. The information infrastructure of today’s Web is not yet capable of providing the levels of repeatability, reliability, accountability and integrity achieved by paper-based scientific information technologies such as books, journals and laboratory notebooks [32]. To demonstrate this point, recall that in September 2008 the share price of United Airlines briefly lost 75% of its value after Google News promoted an undated, six-year-old article concerning United Airlines’ near-bankruptcy in 2002 [1]. We need (but do not

yet have) a *Scientific Web*¹ that reconciles the advantages of fluid electronic data storage, sharing and distributed computation with the concomitant disadvantages.

1.1 Motivating Examples

Scientists are deeply concerned about the quality, integrity and authenticity of their data and conclusions derived from it. If their work turns out to be based on bad data, this can be highly embarrassing or even damaging to their careers [28, 33]. Few scientists would blindly trust the data in a database or in the result of a complicated computation. Instead, such data is expected to be accompanied by *provenance*, or information explaining an object’s history, authorship, derivation, or other information helpful in judging its quality and scientific value. Provenance arises in a number of contexts, including curated databases, workflow management systems and security.

Biological database curation Biological databases on a number of subjects have been developed or are currently under active development. For example, in molecular biology alone there are over 1000 such databases [23]. Each database may incorporate information from a variety of sources, including data entered manually or copied from electronic versions of journal articles and abstracts, data copied from other databases, and data produced by external tools such as BLAST (which themselves may rely on other databases). There is great potential for errors or misinterpretations to proliferate in this complicated data ecosystem. Moreover, it is not even clear how to judge whether the data is correct or incorrect: since the data being collected is the subject of active scientific research, there may be multiple (inconsistent) sources for some result, and so deciding which one to trust requires informed judgment. Biologists find that the best way to ensure that the data is correct (or at least, valid according to the best current science) is for experts to manually enter or correct the data. This process is called *curation* and scientists who construct or maintain such databases are called *curators*.

One major problem with manual curation is its expense in terms of human effort [5]. From a purely economic point of view, it is far more attractive to automate the process of integrating, cleaning, and mining the scientific literature, and there are many well-funded bioinformaticians trying to do exactly that. However, scientists are deeply suspicious of such automation because it may produce results containing untraceable or systematic errors. Artificial intelligence techniques that can provide a 99% success rate might be well and good for commercial applications that are tolerant to a certain amount of risk, but being wrong 1% of the time can wreck a scientist’s career.

Thus, provenance is believed to be essential in scientific data curation for several reasons. First, it records that curation has taken place, ensuring that curation effort is not wasted or duplicated and that scientists can distinguish between manually curated data,

¹By analogy with, and with apologies to, the “Semantic Web”.

raw data, and data synthesized or predicted by some automatic (and otherwise opaque) process. Secondly, provenance provides detailed information about whose judgments informed the selection or correction of the data, and thus who can be held accountable for (or given credit for) these choices.

The behavior of a general-purpose computer system should not be based on subjective or domain-specific assumptions, especially when there is widespread disagreement about the validity of these assumptions, as is frequently the case in cutting-edge science. The best we can do is develop systems that provide *transparency* and *accountability* guarantees concerning the change history, sources, and influences on the data. Such tools should make it possible for users to make effective use of their own domain knowledge rather than imposing some arbitrary measure of data quality.

Scientific workflows Many scientific communities have been pooling computer systems and other resources such as databases and connecting the systems using into Grids that support distributed parallel computation. *workflow management systems* [31, 35] are being developed that provide high-level, graphical notation for programming Grid computations, hiding much of the complexity of implementing a parallel or distributed computation.

Grids are often heterogeneous, may contain multiple different providers for the same service, and may change over time (even within a single long computation). Moreover, some workflow components are based on databases that also change over time. Thus, workflow execution is usually nondeterministic, so that repeated runs of a workflow may yield different results. This is, of course, problematic for a scientist who might wish to use the results of a workflow in a publication, since it undermines the repeatability of an observation, and it is far from clear how to estimate the error arising from nondeterministic workflow enactment.

In Grid or workflow computation, provenance is considered important for several purposes [19]. Provenance helps to identify and re-use results that have already been computed, thus avoiding redundant computations. In addition, since a workflow may be non-deterministic, provenance information is needed to record what actually happened when such a workflow was run, ensuring repeatability. Moreover, in scientific workflows, data provenance can often help the scientist interpret surprising results, i.e., help debug faulty runs (wrong parameter settings, stale data or service bindings, etc), or confirm new findings. Finally, data and computationally-intensive workflows often require runtime monitoring, e.g. to steer the computation, or restart failed subprograms. A great deal of research has already been carried out on understanding provenance in workflow settings, including recent challenge problems and standardization efforts [2]. However, several research challenges remain, particularly understanding how to combine database and workflow provenance [29]

Audit-based security Much research on security (and particularly language-based security [39, 38]) focuses on building models of possible attacks and then designing systems that provably guarantee that attacks are impossible. This approach is quite challenging and although significant progress has been made, we are still far from being able to certify any but the simplest real-world systems or programs secure. Moreover, proofs of security are based on models that may not match the real world, thus may leave the door open to attacks that subvert the model's assumptions. Conversely, the behavior of a provably secure system may be so constrained as to make it useless in practice.

Audit-based security is an alternative approach based on collecting evidence that can be used to detect or react to attacks. *Self-securing storage* is a proposal for computer security based on recording everything that happens to the file system [42, 41, 34]. There are potential security benefits: if an intrusion can be detected

within two weeks, then it suffices to store two weeks' worth of history in order to be able to recover from a typical attack. Another potential application of such provenance information is in aiding information retrieval; for example, helping to locate lost files [40]. Vaughan et al. [43, 30] have studied a language-based audit security framework which combines standard access control techniques with secure logging. Thus, access to a resource may be granted without an explicit security proof but logging is performed to determine whether the access obeys the security policy. This makes it possible to determine whether a resource has been misused and if so, who may be (or is not) responsible. Oracle has also recently introduced a product called *Audit Vault* [17] which provides a logging and log analysis for databases comparable to the above techniques.

For all of these applications there appear to be open research questions. There has been little study of how to make effective use of the logged information (other than to restore the system to a consistent pre-attack state). More importantly, there is so far not a clear understanding of what it means for an audit trail or log to be a correct representation of the system behavior generating it. This is an important issue when one considers that audit information might need to meet high standards of evidence in legal proceedings concerning intellectual property, particularly in disciplines such as biology and pharmacology in which intellectual property concerns are often in direct conflict with scientific concerns.

1.2 The problem

The tools now being used to conduct science on the Web are ill-suited for recording and managing provenance. At present, scientific applications employ a hodgepodge of data models, including flat files, relational databases, and XML, as well as specialized scientific data formats such as FITS. Similarly, a wide variety of programming and scripting languages are used to implement and connect the systems, including Java, C/C++, Perl, and Python.

The wide variety of data models and programming languages currently employed compounds the difficulty of tracking provenance information reliably. This problem arises already even in simple file systems: metadata such as modification and creation timestamps, ownership and permissions is normally discarded or rendered meaningless when files move among different systems. At present, provenance and other metadata is recorded either manually or by custom-built systems. Both approaches are expensive and have additional drawbacks. Manual provenance tracking are not a good uses of scientists' time and are error-prone. Conversely, custom systems can only record provenance within a single system and implement one of a variety of ad hoc or application-specific provenance models. Thus, provenance is still lost when data moves among systems and provenance records from different systems may not be easy to correlate or integrate.

Ideally, provenance would be captured automatically and managed smoothly by all of the components of a scientific application through well-understood techniques based on solid foundations—just as traditional scientific computation is based on the mathematical foundations of real and complex analysis, differential equations, and so on. The current situation falls far short of this ideal. Such tracking would have to involve first agreeing on a protocol for exchanging provenance information whenever two components communicate, and then waiting (and hoping) for all of the tools in use by scientists to implement this protocol. This is impractical in the short term. Instead, most research has focused on understanding the foundations of different models of provenance and incorporating support for various forms of provenance within particular *general-purpose* systems, such as databases, workflow management systems, or file systems. Once provenance management is well-understood for individual systems, we can hope to develop

techniques for integrating provenance management as data moves between systems.

Previous work on provenance in databases [18, 11, 12, 27, 22] has focused on the problem of defining, implementing and studying the properties of different models. Our work in particular [8, 16, 10] emphasizes using techniques from the semantics of programming languages to study provenance, motivated by the observation that previous techniques have sometimes lacked clearly specifications. It is not always clear what a given system actually does, what problem it was intended to solve, or whether it correctly solves it. We believe that formalizing the semantics and desired high-level properties of the various provenance models is a necessary first step towards developing truly scientific provenance management techniques needed for the Scientific Web.

XML plays an essential role in many of these applications and may play a unifying role in the future. Many biological databases are made available online in XML form, even though most are still stored in relational databases. Distributed Grid or workflow computation systems are often based on XML messages and the components of such systems might also be XML databases. Novel lightweight techniques for distributed programming such as MapReduce are typically based on a flat-file or nested-relational data model [20, 36], but could also be generalized to process XML data.

Standardizing on XML data models and databases, programming languages and other tools could help simplify the problem of tracking provenance information when it crosses system boundaries. Moreover, it is of independent interest to understand provenance for XML processors and databases. However, there has been little work on provenance in XML settings (exceptions include [22] and arguably [11, 8]).

Outline In this paper we will adapt some existing models of provenance for relational databases to an XML setting. We first review a core XML query language (Section 2), then define analogous models of provenance for XML data and query languages (Section 3), and discuss the advantages and disadvantages of these models and areas for future work on provenance in scientific applications of XML (Section 4) before concluding (Section 5).

2. Background

We will employ a simple fragment of XQuery in this paper whose semantics can be described easily using a denotational approach (similar to the semantics of (nested) relational query languages). Specifically, we limit attention to the element structure of XML documents, we do not treat node identities, we consider navigation along only the child axis, and we restrict attention to monotone queries. This is, of course, a drastic simplification of XQuery proper, as it ignores the descendant, ancestor, and sibling axes; however, our goal is not to provide a full treatment of provenance for general XML queries, but only to illustrate how existing provenance techniques for (nested) relational data can be extended to XML. This fragment suffices for our purposes.

First, we define values as expressions denoting XML trees, forests and strings. Let Σ be an alphabet of *element tags* a, b and $(\Omega, \epsilon, \cdot, (-)^{-})$ a collection of strings ω equipped with an empty string constant ϵ , concatenation $\omega \cdot \omega'$ and reversal ω^{-} operations. Then (*hedge*) values $v, v' \in Val$ and *tree values* $t, t' \in TVal$ are defined according to the following grammar:

$$v ::= () \mid v, v' \mid t \quad t ::= a[v] \mid \omega$$

where we identify values up to associativity and unit laws: $()$, $v = v = v$, $()$ and (v, v') , $v'' = v, (v', v'')$. The notation $a[v]$ is a concise notation for XML element tags such as $\langle a \rangle \dots \langle /a \rangle$. Note that XQuery processors are typically supposed to normalize the results

of a query so that adjacent strings within a value are concatenated; for example, "foo", "bar" will evaluate to "foobar". We will not model this behavior. We ignore other XML idiosyncrasies such as attributes, comments and processing instructions.

We define some auxiliary functions on tree values:

$$\begin{aligned} \text{select}(b[v], a) &= \begin{cases} b[v] & a = b \\ () & a \neq b \end{cases} \\ \text{select}(\omega, a) &= () \\ \text{children}(a[v]) &= v \\ \text{children}(\omega) &= () \\ \text{text}(\omega) &= \omega \\ \text{text}() &= \epsilon \\ \text{text}(v, v') &= \text{text}(v) \cdot \text{text}(v') \\ \text{text}(a[v]) &= \text{text}(v) \\ \text{concat}(v, v') &= \text{text}(v) \cdot \text{text}(v') \\ \text{reverse}(v) &= \text{text}(v)^{-} \end{aligned}$$

We will employ a *sequence comprehension* operation to define the semantics of for-loops. Such sequence comprehensions are essentially the same as list comprehensions in Haskell, based on using the list monad as a collection type. Formally, this operation is defined as follows:

$$\begin{aligned} \bigsqcup [f(x) \mid x \in ()] &= () \\ \bigsqcup [f(x) \mid x \in t, v'] &= f(t), \bigsqcup [f(x) \mid x \in v'] \end{aligned}$$

Note that this definition is stable with respect to the associativity and unit properties of sequences; for example $\bigsqcup [f(x) \mid x \in (), v] = ()$, $f(v)v = f(v) = \bigsqcup [f(x) \mid x \in v]$.

Now consider countably infinite, disjoint sets $Var = \{x, y, \dots\}$ of *variables* and $TVar = \{\hat{x}, \hat{y}, \dots\}$ of *tree variables*. We define expressions $e \in Expr$ as follows:

$$\begin{aligned} e ::= & x \mid \hat{x} \mid \text{let } x := e_1 \text{ in } e_2 \mid () \mid a[e] \mid e, e' \\ & \mid \hat{x}/\text{child} \mid \hat{x} :: a \mid \text{for } \hat{x} \in e_1 \text{ return } e_2 \\ & \mid \omega \mid x/\text{text} \mid x \cdot y \mid \text{rev}(x) \end{aligned}$$

Variables and let-bindings are standard. The next three expression forms construct values. The \hat{x}/child expression extracts the children of a tree value, $\hat{x} :: a$ filters tree values based on their label, and $\text{for } \hat{x} \in e_1 \text{ return } e_2$ performs iteration over sequences, evaluating $e_2(t)$ for each tree t in e_1 . Finally, we include string constants ω , the \hat{x}/text operator that extracts the text content of a value, the string concatenation operator $x \cdot y$, and the string reversal operator $\text{rev}(x)$ reverses a string (non-strings are implicitly converted to strings first). In some of the provenance models later in the paper, we exclude the string operations.

Remark 1. The difference between ordinary variables and tree variables is that the former may be bound to any value but the latter may only be bound to single trees. Note that tree values are introduced by for and used in \hat{x}/child , $\hat{x} :: n$ and several other expressions.

Our core language excludes general XPath expressions, but these are definable (as long as they involve the child or self axes only). For example we can define XPath expressions such as x/a as $\text{for } \hat{y} \in x \text{ return for } \hat{z} \in \hat{y}/\text{child return } \hat{z} :: a$. Moreover, axis steps involving wildcard ($*$) node tests can be translated by just using child , for example $x/*$ is defined just by $\text{for } \hat{y} \in x \text{ return } \hat{y}/\text{child}$.

We also deliberately exclude conditionals and equality tests to avoid certain complications. XQuery's general equality operator

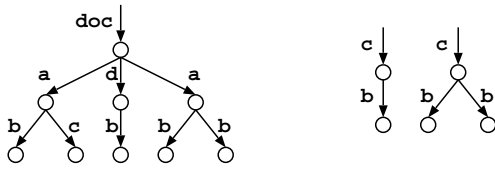


Figure 1. Input and output of query q_1

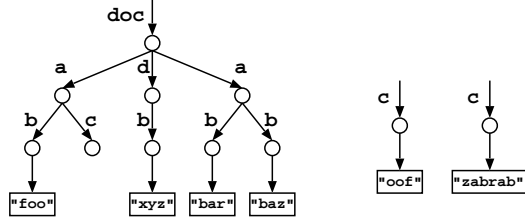


Figure 2. Input and output of query q_2

has complex semantics and can be used to define emptiness tests, which pose difficulties to techniques such as why-provenance.

The string reversal operator is included as a token example of a primitive function on base types.

Semantics We define environments γ as pairs of functions $\gamma_1 : Var \rightarrow Val$ and $\gamma_2 : TVar \rightarrow TVal$; we typically suppress the subscripts. We define the semantics of our core language as follows:

$$\begin{aligned}
\llbracket x \rrbracket \gamma &= \gamma(x) & \llbracket a[e] \rrbracket \gamma &= a[\llbracket e \rrbracket \gamma] \\
\llbracket \hat{x} \rrbracket \gamma &= \gamma(\hat{x}) & \llbracket [e_1, e_2] \rrbracket \gamma &= \llbracket [e_1] \rrbracket \gamma, \llbracket [e_2] \rrbracket \gamma \\
\llbracket () \rrbracket \gamma &= () & \llbracket [\omega] \rrbracket \gamma &= \text{string}(\omega) \\
\llbracket [x/\text{text}] \rrbracket \gamma &= \text{text}(\gamma(x)) \\
\llbracket [x \cdot y] \rrbracket \gamma &= \text{concat}(\gamma(x), \gamma(y)) \\
\llbracket [\text{rev}(x)] \rrbracket \gamma &= \text{reverse}(\gamma(x)) \\
\llbracket [\hat{x}/\text{child}] \rrbracket \gamma &= \text{children}(\gamma(\hat{x})) \\
\llbracket [\hat{x} :: a] \rrbracket \gamma &= \text{select}(\gamma(\hat{x}), a) \\
\llbracket [\text{let } x = e_1 \text{ in } e_2] \rrbracket \gamma &= \llbracket [e_2] \rrbracket (\gamma[x := \llbracket [e_1] \rrbracket \gamma]) \\
\llbracket [\text{for } \hat{x} \in e_1 \text{ return } e_2] \rrbracket \gamma &= \bigsqcup \llbracket [e_2] \rrbracket [\hat{x} := t] \mid t \in \llbracket [e_1] \rrbracket \gamma
\end{aligned}$$

Example 1. We will introduce a running examples for use throughout the paper. Consider the query

$$q_1 = \text{for } y \in x/a \text{ return } c[y/b]$$

which, as discussed earlier, can be represented in our core language as follows:

$$\begin{aligned}
&\text{for } \hat{y}' \in x/\text{child} \text{ return for } \hat{y} \in \hat{y}' :: a \text{ return} \\
&c[\text{for } \hat{z} \in \hat{y}/\text{child} \text{ return } \hat{z} :: b]
\end{aligned}$$

Figure 1 shows a typical input and output for this query.

Similarly, consider the query

$$q_2 = \text{for } y \in x/a \text{ return } c[\text{rev}(y)].$$

This query gets the text content of each a node below the root, reverses it, and returns the reversed string in a c node. Figure 2 shows a typical input and output for this query.

3. Models of Provenance

In this section we will define several models of provenance. Each model fits into a general framework as we shall now explain. For a given model of provenance \mathcal{P} , we shall specify:

1. Define \mathcal{P} -annotated XML values (or just *pp-values*) $Val^{\mathcal{P}}$ (along with \mathcal{P} -tree values $TVal^{\mathcal{P}}$ and \mathcal{P} -environments mapping variables to \mathcal{P} -values.)
2. Define an annotation-propagating variant $\mathcal{P}[\llbracket - \rrbracket]$ of the standard XQuery semantics on \mathcal{P} -values
3. Consider the provenance semantics to be the behavior of an expression on certain *distinctly annotated* inputs
4. State and prove well-behavedness properties and perhaps additional behavioral guarantees

Although there are a number of other possible ways to model provenance and define provenance semantics, we have (through bitter experience) found this approach to be the best suited to understanding and analysis of such models. Some work on provenance has employed ad hoc, noncompositional, or incomplete definitions, making it difficult to understand and relate the models, let alone formalize and prove their properties. While our approach may not seem natural at first sight (especially to readers without past experience with programming language semantics), every provenance model we are aware of can be defined (often much more clearly) in this form. Moreover, in this framework we can define some of the high-level design principles we will follow so that we do not have to repeatedly discuss them.

Validity The most basic correctness property we are interested in is that the annotation-propagating semantics is faithful to the standard semantics. To formalize this we assume (as shall generally be the case) that there is an obvious *erasure* function $| - | : Val^{\mathcal{P}} \rightarrow Val$ mapping annotated values to ordinary ones.

Definition 1 (Validity). An annotation-propagating semantics is *valid* provided $\mathcal{P}[\llbracket e \rrbracket]v = | \mathcal{P}[\llbracket e \rrbracket]v |$.

All of the definitions we will give are valid and we will not mention or prove this explicitly.

Invariance Another typical property is that the annotation-propagation semantics is invariant in a certain sense. Specifically, the particular choice of annotations should not matter. We consider a reannotation function $\alpha : Val^{\mathcal{P}} \rightarrow Val^{\mathcal{P}}$ to be a function on annotated values such that $|\alpha(v)| = |v|$.

Definition 2 (Invariance). An annotation-propagating semantics is *invariant* under some class of reannotations T provided for all $\alpha \in T$, we have $\mathcal{P}[\llbracket e \rrbracket](\alpha(v)) = \alpha(\mathcal{P}[\llbracket e \rrbracket]v)$.

Invariance properties are familiar to both database theorists (as (full) genericity on annotations [6, 7]) and programming language theorists (as parametricity in polymorphic languages [37], if we view annotations as an abstract type). All of the techniques we will consider are invariant with respect to appropriate classes of reannotations but we will not further discuss these invariance properties.

Compositionality Each of our models can be defined *compositionally*. This simply means (informally) that the semantics can be defined by giving functions that show how to handle each expression form individually.

Definition 3. Given \mathcal{P} -annotated values $Val^{\mathcal{P}}$, let $\mathcal{P}[\llbracket - \rrbracket]$ be a function such that $\mathcal{P}[\llbracket e \rrbracket]$ maps \mathcal{P} -environments to \mathcal{P} -values for each e . We say that $\mathcal{P}[\llbracket - \rrbracket]$ is *compositional* if there exist generating func-

tions

$$\begin{aligned}
\text{emp}^{\mathcal{P}} &: \text{Val}^{\mathcal{P}} \\
\text{seq}^{\mathcal{P}} &: \text{Val}^{\mathcal{P}} \times \text{Val}^{\mathcal{P}} \rightarrow \text{Val}^{\mathcal{P}} \\
\text{elt}^{\mathcal{P}} &: \Sigma \times \text{Val}^{\mathcal{P}} \rightarrow \text{Val}^{\mathcal{P}} \\
\text{string}^{\mathcal{P}} &: \Omega \rightarrow \text{Val}^{\mathcal{P}} \\
\text{concat}^{\mathcal{P}} &: \text{Val}^{\mathcal{P}} \times \text{Val}^{\mathcal{P}} \rightarrow \text{Val}^{\mathcal{P}} \\
\text{reverse}^{\mathcal{P}} &: \text{Val}^{\mathcal{P}} \rightarrow \text{Val}^{\mathcal{P}} \\
\text{text}^{\mathcal{P}} &: \text{Val}^{\mathcal{P}} \rightarrow \text{Val}^{\mathcal{P}} \\
\text{children}^{\mathcal{P}} &: T\text{Val}^{\mathcal{P}} \rightarrow \text{Val}^{\mathcal{P}} \\
\text{select}^{\mathcal{P}} &: T\text{Val}^{\mathcal{P}} \times \Sigma \rightarrow \text{Val}^{\mathcal{P}} \\
\text{for}^{\mathcal{P}} &: \text{Val}^{\mathcal{P}} \times (T\text{Val}^{\mathcal{P}} \rightarrow \text{Val}^{\mathcal{P}}) \rightarrow \text{Val}^{\mathcal{P}}
\end{aligned}$$

such that:

$$\begin{aligned}
\mathcal{P}[x]\gamma &= \gamma(x) & \mathcal{P}[a[e]]\gamma &= \text{elt}^{\mathcal{P}}(a, \mathcal{P}[e]\gamma) \\
\mathcal{P}[\hat{x}]\gamma &= \gamma(\hat{x}) & \mathcal{P}[e_1, e_2]\gamma &= \text{seq}^{\mathcal{P}}(\mathcal{P}[e_1]\gamma, \mathcal{P}[e_2]\gamma) \\
\mathcal{P}[\omega]\gamma &= \text{emp}^{\mathcal{P}} & \mathcal{P}[\omega]\gamma &= \text{string}^{\mathcal{P}}(\omega) \\
\mathcal{P}[x/\text{text}]\gamma &= \text{text}^{\mathcal{P}}(\gamma(x)) \\
\mathcal{P}[\text{rev}(x)]\gamma &= \text{reverse}^{\mathcal{P}}(\gamma(x)) \\
\mathcal{P}[x \cdot y]\gamma &= \text{concat}^{\mathcal{P}}(\gamma(x), \gamma(y)) \\
\mathcal{P}[\hat{x}/\text{child}]\gamma &= \text{children}^{\mathcal{P}}(\gamma(\hat{x})) \\
\mathcal{P}[\hat{x} :: a]\gamma &= \text{select}^{\mathcal{P}}(\gamma(\hat{x}), a) \\
\mathcal{P}[\text{let } x := e_1 \text{ in } e_2]\gamma &= \mathcal{P}[e_2](\gamma[x := \mathcal{P}[e_1]\gamma]) \\
\mathcal{P}[\text{for } \hat{x} \in e_1 \text{ return } e_2]\gamma &= \text{for}^{\mathcal{P}}(\mathcal{P}[e_1]\gamma, \Lambda t. \mathcal{P}[e_2]\gamma[\hat{x} := t])
\end{aligned}$$

The standard semantics is clearly compositional, with:

$$\begin{aligned}
\text{emp} &= () & \text{concat}(v, v') &= \text{concat}(v, v') \\
\text{seq}(v_1, v_2) &= v_1, v_2 & \text{reverse}(v) &= \text{reverse}(v) \\
\text{elt}(a, v) &= a[v] & \text{children}(t) &= \text{children}(t) \\
\text{string}(\omega) &= \omega & \text{select}(t, a) &= \text{select}(t, a) \\
\text{text}(v) &= \text{text}(v) & \text{for}(v, f) &= \bigsqcup [f(t) \mid t \in v]
\end{aligned}$$

In practice, compositionality seems essential for obtaining a definition that is well-behaved enough for simple properties to have simple proofs. In particular, note that checking validity and invariance for compositional definitions reduces almost immediately to checking the corresponding properties for the generating functions.

3.1 Where-Provenance

Where-provenance, as introduced in [11, 12], is information linking data in the result of an operation showing “where the output data came from”. As discussed by Buneman et al. [10], a key property of where-provenance is that it links parts of the output to equal parts of the input and moreover that the links reflect the true behavior of the query.

In an XML/XQuery setting, we can model where-provenance as follows. Consider \mathcal{W} -values in which each subtree is tagged with extra information α . We refer to the tags as *colors*, and identify a constant, *blank* color \perp .

$$v ::= () \mid v, v' \mid t \quad t ::= a[v]^\alpha \mid s^\alpha \quad \alpha ::= \perp \mid c$$

The idea of our approach to defining where-provenance is as follows: First, we consider a value whose colors are all distinct. We then define a semantics $\mathcal{W}[e]$ of expressions acting on colored values that propagates colors from the input to the output. By running $\mathcal{W}[e]$ on a distinctly-colored value, we obtain a value annotated with colors that can be interpreted as links to unique parts of the distinctly colored input.

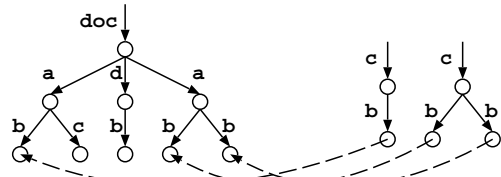


Figure 3. Where-provenance behavior of query q_1

The annotation-propagation semantics for where-provenance $\mathcal{W}[-]$ can be specified by the generating functions:

$$\begin{aligned}
\text{elt}^{\mathcal{W}}(a, v) &= a[v]^\perp \\
\text{text}^{\mathcal{W}}(v) &= \text{text}(v)^\perp \\
\text{concat}^{\mathcal{W}}(v, v') &= \text{concat}(v, v')^\perp \\
\text{reverse}^{\mathcal{W}}(v) &= \text{reverse}(v)^\perp \\
\text{children}^{\mathcal{W}}(a[v]^\alpha) &= v \\
\text{children}^{\mathcal{W}}(\omega^\alpha) &= () \\
\text{select}^{\mathcal{W}}(a[v]^\alpha, b) &= \begin{cases} a[v]^\alpha & a = b \\ () & a \neq b \end{cases} \\
\text{select}^{\mathcal{W}}(\omega^\alpha, b) &= () \\
\text{for}^{\mathcal{W}}(v, f) &= \bigsqcup [f(t^\alpha) \mid t^\alpha \in v]
\end{aligned}$$

The remaining generating functions are standard. Moreover, it is not hard to show that this semantics is valid, invariant up to recolorings, and it is compositional by definition.

Example 2. Figure 3 shows the where-provenance semantics of query q_1 (from Example 1), where we use links drawn using dotted lines to connect annotated parts of the output to their unique sources in the input. We do not show q_2 since its where-provenance semantics is uninteresting (every part of the output is labeled \perp).

Correctness We now motivate and state the correctness property we have in mind for where-provenance. Imagine we have a distinctly-annotated value $v = a[b]^\beta, b]^\gamma]^\alpha$. The distinct annotations α, β, γ serve as “addresses” for the three subtrees of b ; in particular, β and γ distinguish the two occurrences of b . Now imagine we have computed some value v' from v , for example $d[b]^\beta, c]^\gamma]^\perp$. Here, if we interpret the annotations as “pointers” back into v , then the first subtree $b]^\beta$ is consistent with v since β labels the same subtree b in v , but the second subtree $c]^\gamma$ is not. On the other hand, if we have a result $v'' = d[a[b]^\beta, b]^\gamma]^\alpha, b]^\gamma]^\perp$ then this is indeed consistent with v .

The basic observation here is that if we are interpreting annotations as provenance links to “sources” in the data, then we expect the copy to be similar, if not identical, to the source. To formalize this notion we need another bit of notation.

Definition 4 (Colored subtrees and copying). We define the *colored subtrees* of v as follows:

$$\begin{aligned}
\text{cst}() &= \emptyset & \text{cst}(v, v') &= \text{cst}(v) \cup \text{cst}(v') \\
\text{cst}(a[v]^\alpha) &= \{a[v]^\alpha\} \cup \text{cst}(v)
\end{aligned}$$

Moreover, we write $\text{cst}^\neq(v)$ for the non- \perp -colored subtrees of v , that is $\{t[v]^\alpha \in \text{cst}(v) \mid \alpha \neq \perp\}$. A function $F: \text{Val}^{\mathcal{W}} \rightarrow \text{Val}^{\mathcal{W}}$ is *copying* if for every v , we have $\text{cst}^\neq(F(v)) \subseteq \text{cst}(v)$.

The where-provenance semantics given above is guaranteed to be copying:

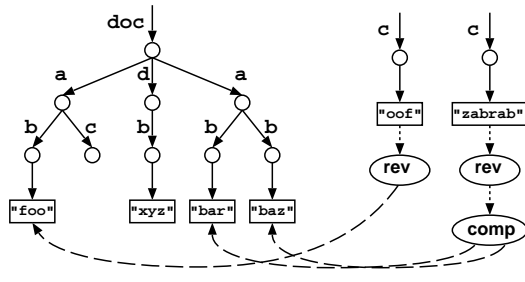


Figure 4. Expression-provenance for query q_2

Theorem 1. For any e , the where-provenance semantics $\mathcal{W}[[e]]$ is copying.

The proof of this property is similar to that for the NRC where-provenance model given in [10]. Moreover, this implies that for distinctly-colored inputs, each non- \perp -colored part of the output is a copy of the part of the input with the same color.

3.2 Expression-Provenance

The simple where-provenance model above only gives us useful information about parts of the output are exact copies of parts of the input. This is often not the case, particularly in the presence of base types (strings, integers) and primitive functions such as `rev`, or external function calls. Above, we followed the obvious strategy of coloring the results of `rev` or `text` operations with \perp . However, this obviously throws away some useful information. One idea (developed recently with Vansummeren, and partly inspired by the semiring relational model [27]) is to annotate values of base type with *expressions*:

$$v ::= () \mid v, v' \mid t \quad t ::= a[v] \mid \omega^\alpha \quad \alpha ::= c \mid \omega \mid \alpha \cdot \beta \mid \alpha^-$$

To simplify matters we will only consider annotations on strings here, but it is possible to combine these annotations with the where-provenance semantics in the previous section.

The expression-provenance semantics $\mathcal{E}[[_]]$ can now be defined compositionally using the generating functions:

$$\begin{aligned} \text{string}^\mathcal{E}(\omega)\gamma &= \omega^\omega \\ \text{concat}^\mathcal{E}(\omega_1^\alpha, \omega_2^\beta) &= (\omega_1 \cdot \omega_2)^{\alpha \cdot \beta} \\ \text{text}^\mathcal{E}(\omega^\alpha) &= \omega^\alpha \\ \text{text}^\mathcal{E}(())) &= \epsilon \\ \text{text}^\mathcal{E}(v, v') &= \text{concat}^\mathcal{E}(\text{text}^\mathcal{E}(v), \text{text}^\mathcal{E}(v')) \\ \text{text}^\mathcal{E}(a[v]) &= \text{text}^\mathcal{E}(v) \\ \text{reverse}^\mathcal{E}(\omega^\alpha) &= (\text{reverse}(\omega))^{\alpha^-} \\ \text{reverse}^\mathcal{E}(a[v]) &= \text{reverse}(\text{text}(a[v])) \end{aligned}$$

where as usual the omitted generating functions are standard.

Example 3. The expression-provenance of q_1 is essentially the same as its where-provenance, since no strings are involved. Figure 4 shows the expression-provenance of query q_2 that traverses a document and returns the text content under each a node packed into a new c node. Here, we show the expressions as explicit syntax trees. For example, the annotation on "zabrab" is $(\beta \cdot \delta)^-$ provided β and δ are the annotations of "bar" and "baz" respectively.

Correctness By analogy with the correctness property of where-provenance we expect the annotations of output parts to be “consistent” with the corresponding input parts. Since annotations now have more structure, we need to take that structure into account. We formalize this as follows.

Consider a function $h : \text{Color} \rightarrow \Omega$ which assigns a string to each color. We call this a valuation. Given an annotation α , we can extend h to map annotations α to strings as follows:

$$\begin{aligned} h[c] &= h(c) & h[\alpha \cdot \beta] &= h[\alpha] \cdot h[\beta] \\ h[s] &= s & h[\alpha^-] &= h[\alpha]^- \end{aligned}$$

Definition 5. We define the colored substrings of an \mathcal{E} -annotated value $\text{css}(v)$ as the set of all annotated strings s^α occurring in v , just as in Definition 4. We say an \mathcal{E} -value v is *consistent* with a valuation h , written $h \approx v$, as follows.

$$h \approx v \iff \forall s^\alpha \in \text{css}(v). h[\alpha] = s.$$

These notions are extended to tree values and environments in the obvious way.

Given a distinctly annotated input in which each string is tagged with a unique color, it is trivial to find a consistent valuation h . Moreover, it is not hard to show that the expression-provenance semantics preserves consistency in the following sense.

Definition 6. A function f on \mathcal{E} -values is *consistency-preserving* if for any $h : \text{Color} \rightarrow \Omega$ and v satisfying $h \approx v$, we have $h \approx F(v)$.

Theorem 2. The expression-provenance semantics $\mathcal{E}[[e]]$ is consistency-preserving for any e .

3.3 Why-Provenance

Why-provenance, and its cousin *lineage* [18], are based on the intuition of identifying the properties of the input that “witness” or “justify” a certain part of the output. In a relational database setting, one can formulate this in a straightforward way: a witness to a record t in the output of a query Q is a subset J of the input database I that satisfies $t \in Q(J)$. Witnesses are not as well-behaved in the presence of string operations, so in this section we will consider the sublanguage without strings, `text` or `rev` operations.

Buneman, Khanna and Tan [11, 12] studied why-provenance for a deterministic tree model and for flat relational databases. Later, Green et al. [27] introduced a *semiring-valued* relational model and argued that why-provenance and lineage can be obtained as an instance of this model. The semiring model has been refined in subsequent work [9, 24, 26]; moreover, Foster et al. [22] extended the semiring-valued model to the nested-relational calculus and an XQuery-like language over unordered XML trees. These papers should be consulted for more detail on the semiring-valued model; we will not go into further details in this paper.

For *ordered* XML, however, the definition of why-provenance is less clear. The semiring model does not apply to ordered collections; thus, in a setting where element order is important, the semiring model and accompanying results cannot be used. Nevertheless, we will consider an approach to why-provenance for ordered XML that draws on the idea of witness.

First, to make the analogy to relational why-provenance more precise, we introduce an “information ordering” \sqsubseteq on XML trees.

$$\frac{}{() \sqsubseteq v} \quad \frac{v_1 \sqsubseteq v'_1 \quad v_2 \sqsubseteq v'_2}{v_1, v_2 \sqsubseteq v'_1, v'_2} \quad \frac{v \sqsubseteq v'}{a[v] \sqsubseteq a[v']}$$

Intuitively, the idea is that $x \sqsubseteq y$ means that x can be obtained from y by deleting some subtrees; alternatively, that y has at least as much information as x . This generalizes both the idea of containment of relational databases and membership of a record in a relational database. (Indeed, this idea was already used in [11] to define why-provenance for a semistructured data model).

Using the information order, we can define witnesses for XML queries as follows:

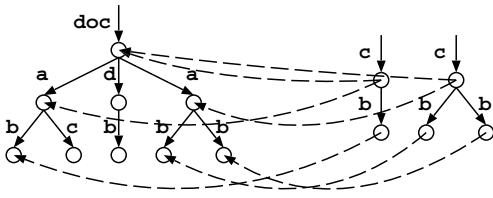


Figure 5. Why-provenance behavior of query q_1

Definition 7. Suppose e, γ and v are given and $v \sqsubseteq \llbracket e \rrbracket \gamma$. Then $w \sqsubseteq \gamma$ is a *witness* to $v \sqsubseteq \llbracket e \rrbracket \gamma$ provided $w \sqsubseteq \llbracket e \rrbracket \gamma$. We define $\text{Wit}(e, \gamma, v)$ as the set of all witnesses to $w \in \llbracket e \rrbracket \gamma$, that is

$$\text{Wit}(e, \gamma, v) = \{w \sqsubseteq \gamma \mid v \sqsubseteq \llbracket e \rrbracket \gamma\}.$$

The intuition is that a witness provides enough information to “force” a subtree to appear in the output. Moreover, note that t and w may be arbitrary trees.

Now, to compute why-provenance, we again consider annotating each part of a value, but we now consider annotations to be sets of “colors” rather than just single colors. Initially, the annotations in distinctly colored value will be singletons with no colors re-used.

$$v ::= () \mid v, v' \mid \alpha : t \quad t ::= a[v]$$

where $\alpha \subseteq \text{Color}$. Note that this grammar is essentially the same as that for where-provenance annotations but for technical reasons it makes more sense to put the annotations in the value syntax here. We also will need an auxiliary annotation merging operation $v^{+\alpha}$ defined as follows:

$$\begin{aligned} ()^{+\alpha} &= () & (v_1, v_2)^{+\alpha} &= v_1^{+\alpha}, v_2^{+\alpha} \\ (\beta : t)^{+\alpha} &= \alpha \cup \beta : t \end{aligned}$$

We define the why-provenance semantics for XQuery $\mathcal{Y}[\llbracket - \rrbracket]$ with the following (non-standard) generating functions:

$$\text{elt}^{\mathcal{Y}}(a, v) = \emptyset : a[v] \quad \text{for}^{\mathcal{Y}}(v, f) = \bigsqcup \llbracket f(t)^{+\alpha} \mid \alpha : t \in v \rrbracket$$

Example 4. Figure 5 shows the why-provenance behavior of query q_1 . The links drawn with dotted lines connect each output node with the set of input nodes that form the why-provenance of the output node, based on running $\mathcal{Y}[\llbracket q_1 \rrbracket]$ on a distinctly annotated input. We omit q_2 since we do not consider why-provenance in the presence of string operations.

Correctness As motivated above, we want the why-provenance of a part of the output of a query to tell us something about its witnesses. For a distinctly annotated value we can describe a part of the value as a set of labels. To formalize this idea, we define the *restriction operator*:

$$\begin{aligned} ()|_{\alpha} &= () & (v_1, v_2)|_{\alpha} &= v_1|_{\alpha}, v_2|_{\alpha} \\ (a[v])|_{\alpha} &= a[v|_{\alpha}] & (\beta : t)|_{\alpha} &= \begin{cases} t|_{\alpha} & \beta \subseteq \alpha \\ () & \beta \not\subseteq \alpha \end{cases} \end{aligned}$$

A crucial point here is that the restriction of a tree t is always still a tree t' . This is the reason why we place the annotations in the syntax of values: it is sensible to replace the value $\beta : t$ with $()$ but it is not sensible to replace a tree value t with an empty sequence (since the empty sequence is not a tree value).

Proposition 1. *If γ is a valid \mathcal{Y} -environment then $\gamma|_{\alpha}$ is a valid environment.*

We now define a function which extracts potential witnesses from the why-provenance annotations generated by the semantics above.

Definition 8. We define the *provenance witnesses* of a query e with respect to input γ and output part v as:

$$\text{PWit}(e, \gamma, v) = \{\gamma|_{\alpha} \mid v \sqsubseteq (\mathcal{Y}[\llbracket e \rrbracket \gamma])|_{\alpha}\}$$

Intuitively, $\text{PWit}(e, \gamma, v)$ examines all of the parts of the output of the form $(\mathcal{Y}[\llbracket e \rrbracket \gamma])|_{\alpha}$ that contain v , and collects (potential) witnesses by restricting γ according to α . In fact, this set can be quite large since we do not restrict attention to “minimal” annotation sets α having this property; doing so would likely be more efficient but we ignore the efficiency issue for now. Instead we just wish to show that these are all actually witnesses. The main step in doing so is showing that restriction satisfies a natural commutativity property:

Theorem 3. *Let e, γ and α be given. Then $(\mathcal{Y}[\llbracket e \rrbracket \gamma])|_{\alpha} = \llbracket e \rrbracket \gamma|_{\alpha}$.*

Definition 9. A function f on \mathcal{Y} -annotated values is *witnessing* if for any distinctly-annotated γ and any $w \sqsubseteq f(\gamma)$, we have $\text{PWit}(e, \gamma, v) \subseteq \text{Wit}(\llbracket f \rrbracket, \llbracket \gamma \rrbracket, v)$

Corollary 1. *For any e , the why-provenance semantics $\mathcal{Y}[\llbracket e \rrbracket]$ is witnessing.*

Proof. If $\gamma|_{\alpha} \in \text{PWit}(e, \gamma, v)$ then $v \sqsubseteq (\mathcal{Y}[\llbracket e \rrbracket \gamma])|_{\alpha} = \llbracket e \rrbracket \gamma|_{\alpha}$. \square

3.4 Dependency-Provenance

Cheney, Ahmed and Acar [16] introduced another form of provenance called *dependency-provenance*. The motivation for this form of provenance is to capture “dependence” information (analogous to techniques for information-flow security, program slicing and dependency analysis [4, 3]). From a technical point of view, the difference between why-provenance and dependency-provenance is that the former considers only deletion (recall that $v \sqsubseteq w$ means v can be obtained by deleting subtrees from w) whereas the latter considers arbitrary changes to the input. Thus, in general dependency provenance will generally include why-provenance since it is sensitive to more kinds of changes. Moreover, dependency-provenance can handle string operations without difficulty.

For dependency-provenance, we annotate values with sets of colors just as for why-provenance, but we allow annotations to appear anywhere in a value:

$$v ::= () \mid v, v' \mid t \mid \alpha : v \quad t ::= a[v] \mid w$$

We implicitly treat $\alpha : \beta : v$ and $\alpha \cup \beta : v$ as equivalent.

The dependency-provenance semantics $\mathcal{D}[\llbracket - \rrbracket]$ is defined using the following generating functions (standard cases omitted):

$$\begin{aligned} \text{reverse}^{\mathcal{D}}(v) &= \alpha : \omega^{-} \quad (\text{where } \text{text}^{\mathcal{D}}(v) = \alpha : \omega) \\ \text{concat}^{\mathcal{D}}(v_1, v_2) &= \alpha_1 \cup \alpha_2 : \omega_1 \cdot \omega_2 \\ &\quad (\text{where } \text{text}^{\mathcal{D}}(v_i) = \alpha_i : \omega_i) \\ \text{text}^{\mathcal{D}}(\alpha : v) &= \alpha : \text{text}^{\mathcal{D}}(v) \\ \text{for}^{\mathcal{D}}(\alpha : v, f) &= \alpha : \text{for}^{\mathcal{D}}(v, f) \end{aligned}$$

Specifically, a `for`-loop is evaluated by traversing the value to search for tree values, lifting any tags encountered.

Example 5. Figure 6 shows the dependency-provenance behavior of q_1 , and Figure 7 shows the dependency-provenance of q_2 . There are some parts of the output that carry annotations that are not directly associated with a node. These parts are drawn using boxes. Moreover, we also include links from empty subsequences in the output that carry an annotation, such as $\alpha : ()$. These parts are shown as boxes with dotted lines from their parent nodes. These parts were empty in this run of the query, but could become nonempty if the input is changed at α . This information is discarded in why-provenance.

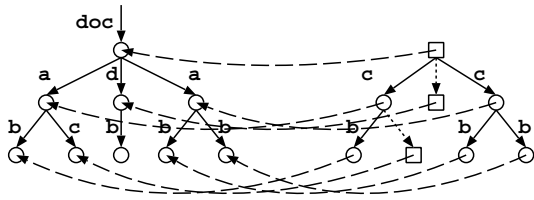


Figure 6. Dependency-provenance behavior of query q_1

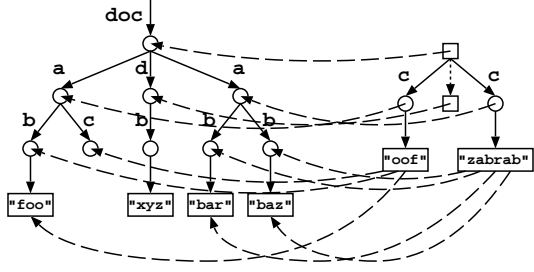


Figure 7. Dependency-provenance behavior of query q_2

Correctness To show the correctness of dependency-provenance, we introduce the following relation called *equivalence above α* :

$$\frac{}{() \equiv_{\alpha} ()} \quad \frac{v_1 \equiv_{\alpha} v'_1 \quad v_2 \equiv_{\alpha} v'_2}{v_1, v_2 \equiv_{\alpha} v'_1, v'_2} \quad \frac{v \equiv_{\alpha} v'}{a[v] \equiv_{\alpha} a[v']} \quad \frac{}{\omega \equiv_{\alpha} \omega}$$

$$\frac{v \equiv_{\alpha} \omega}{\beta : v \equiv_{\alpha} \beta : \omega} \quad \frac{\alpha \subseteq \beta \cap \gamma}{\beta : v \equiv_{\alpha} \gamma : \omega}$$

Intuitively, the idea is that $v \equiv_{\alpha} v'$ holds if v and v' look the same if we ignore subtrees whose respective annotations contain α . Typically, α is just a singleton $\{a\}$, and if v is distinctly-colored then $v \equiv_a v'$ just means that v is the same as v' everywhere except at the subtree of v labeled with a .

Definition 10. A function f on \mathcal{D} -values is *dependency-preserving* if for any α, v, v' if $v \equiv_{\alpha} v'$ then we have $f(v) \equiv_{\alpha} f(v')$.

The intuition here is that the colors ought to capture all of the possible ways the output could change as a result of a change to the input. As illustrated in the example above, in general this requires keeping track of more annotations than why-provenance, because dependency-preservation is sensitive to arbitrary changes, not just deletion. This is closely related to the idea of noninterference in information-flow security [38, 3]. See [16] for further discussion of dependency-correctness and noninterference.

Theorem 4. For any e , the dependency-provenance semantics $\mathcal{D}[e]$ is dependency-preserving.

As usual this can be proved by induction on the structure of expressions, after establishing a number of dependency-preservation properties for the generating functions of \mathcal{D} .

4. Discussion

4.1 Loose specifications

In the last section, we introduced several provenance models that fit into a common framework based on the properties of validity, invariance, and compositionality. These properties represent basic expectations about what it means for a provenance or annotation semantics to be well-behaved. In addition, for each semantics we identified a formal correctness property that we believe can be used to explain the expected behavior of the system to users. However,

in these properties may not enough by themselves to uniquely determine what the system should do.

Where-provenance Consider an alternative where-provenance semantics that runs the query as usual and then simply discards all of the colors. This semantics is copying since the copying property says nothing about the behavior of \perp -colored outputs. In other words, we have not specified that the where-provenance should be “nontrivial” in this way, and it is not obvious how to do so. In fact, Buneman et al. [10] goes further than soundness: they established a converse, *expressive completeness* result showing, roughly, that all copying operations (in a suitably enriched query language) can be expressed by the where-provenance semantics. It is reasonable to expect a similar expressive completeness result to hold for XML where-provenance, but we will leave this as a subject for future work.

Expression-provenance For expression-provenance, the problem is similar to that of where-provenance. Specifically, we can define an alternative semantics that annotates each data value in the result with a copy of itself. For example, if the expression is $x + y$, the evaluating with $x = s, y = t$ will yield $(s \cdot t)^{s \cdot t}$ instead of $st^{x \cdot y}$. This is still consistency-preserving, yet clearly trivial in some sense. As with where-provenance, it is not obvious how to enforce nontriviality. We are aware of preliminary work by Vansummeren² on expressive completeness for expression-provenance of this form, but there are several potential obstacles.

Why-provenance For why-provenance, imagine an alternative definition that tags each part of the output with *every* color in the input. This semantics would also be witnessing, but yields no (useful) information about the witnesses of the query, since it simply says that the entire input witnesses each part of the output. For relational why-provenance, the provenance witnesses “represent” the set of all witnesses in a suitable sense. Moreover, there is a unique *minimal witness basis* that represents all of the witnesses in the most concise possible way. It would be worthwhile to study comparable completeness or minimality properties.

Dependency-provenance For dependency-provenance, there is again a trivial alternative dependency-correct semantics that lifts all of the input colors to the root of the output. Like why-provenance, dependency-provenance is more informative when fewer annotations are present in the output. In our earlier work [16] we studied the issue of minimality of dependency-provenance. This problem turns out to be undecidable for full (nested) relational queries, and intractable even for small sublanguages (e.g. just Boolean expressions). Since XQuery and nested relational queries are closely related, we expect similar reductions to hold for the XML dependency-provenance model. Nevertheless, it may be interesting to consider whether further constraints (such as compositionality) would suffice to uniquely characterize the dependency-provenance semantics.

4.2 Language extensions

The fragment of XQuery we considered leaves out many features, including base types and primitive functions (e.g. booleans, integers), conditionals, emptiness and path-existence tests, recursion, most of the XPath axes, and update operations.

Other base types and primitive functions We used strings with concatenation and reversal as an example of a base type and associated operations. Integers, booleans, arithmetic and propositional logic operations can be added following the same pattern to most of the models we considered (except why-provenance). A more

²Personal communication, May 2008

interesting question is how to handle the union, difference, sorting, grouping and aggregation operations that are present in full XQuery. For example, expression provenance deals only with annotations on strings, but in the presence of aggregation operators such as `sum`, we might also need to refer to the results of queries that construct trees or sequences, not just strings.

Conditionals, emptiness and path tests XQuery includes conditionals that allow testing for the existence of paths or testing whether an expression is empty. Path existence tests by themselves do not pose a major difficulty for provenance tracking. However, why-provenance is not yet completely well-understood for query languages that include negation or emptiness tests. Geerts and Poggi [24] have investigated this issue using the semiring relational model. The other forms of provenance studied here do not appear sensitive to emptiness tests.

Functions and recursion Databases may allow local function definitions but seldom allow first-class lambda-abstraction; however, XQuery allows arbitrary recursive function definitions. These features can be accommodated easily by interpreting lambda-abstraction and application expressions in the standard way:

$$\begin{aligned} \mathcal{P}[\lambda x.e]\gamma &= \Lambda v.\mathcal{P}[e]\gamma[x := v] \\ \mathcal{P}[e_1 e_2]\gamma &= (\mathcal{P}[e_1]\gamma) (\mathcal{P}[e_2]\gamma) \end{aligned}$$

Recursion can also be handled in the standard way, by adjusting the semantics to allow for the possibility of nontermination. Note that the above approach interprets function expressions as functions from annotated values to annotated values; function values themselves do not carry annotations, and so we will not be able to distinguish between, say, $(\lambda x.b[x], c[x]) a[]$ and $b[a[]], c[a[]]$. It may be interesting to allow annotations on functions themselves, in which case we would need to allow nonstandard (but compositional) interpretations of lambda-abstraction and application expressions.

Other XPath axes Full XPath includes additional “downward” axes (descendant, attribute), “upwards” axes (ancestor, parent) and “sibling” axes (previous-sibling, next-sibling). Our core language only supports the child and (trivial) self axis. The other downward axes can easily be accommodated by adding attributes to the data model and using suitable recursive definitions for the descendant axis under various semantics. The other directions are more challenging, because we cannot even define the ordinary semantics of expressions purely in terms of values in the presence of the upwards or sideways axes. The formal semantics of XQuery uses a “store” containing node identifiers [21]. We believe it is possible to extend the approach of this paper to a more realistic XQuery fragment based on stores and node identifiers, but doing so seems to require much heavier notation (e.g. relations rather than functions).

Updates As discussed in the introduction, a major motivation for studying provenance is that curated databases are updated frequently. Here, we have focused on provenance semantics for queries only. Understanding provenance for queries is a necessary first step towards understanding provenance for general XML transformations; moreover, query provenance models can be used to propagate information about the freshness or update history of the underlying data used by a query. However, it is also important to study provenance for update operations [8]. Buneman et al. [10] also studied where-provenance for an NRC-based update language. Similar developments ought to be possible for the XML where-provenance model, possibly using the FLUX update language [14, 15]. However, FLUX is much simpler than the update languages typically encountered in practice such as the W3C XQuery Update Facility proposal [13].

Additional issues As discussed by Buneman et al. [10], there are a number of implementation challenges for provenance and anno-

tation in database systems, including choice of concrete data structures and query optimization. Moreover, we have only addressed the problem of defining and modeling provenance, not on making use of provenance or other annotations, and this raises several interesting language-design questions.

4.3 Is this what scientists need or want?

In the introduction, we presented a wide variety of motivations and real-world concerns that provenance is meant to address. However, we have restricted attention to the subproblem of defining and tracking provenance through XQuery-style queries over XML documents. Already in this simple setting, we encountered some subtle issues and design decisions. Nevertheless, we should consider whether this work addresses the needs of scientific users that we have claimed as motivation. This is tricky to evaluate for several reasons. First, different disciplines have different needs. Second, scientists are busy and typically do not have time to participate in rigorous usability studies.

Nevertheless, we believe that developing concrete, small and easily understood proposals for provenance-tracking techniques is worthwhile because we can use them to get a sense of what properties users consider important or irrelevant. It seems clear that there is a wide range of provenance models with significant variation in detail and performance overhead. So, identifying aspects of provenance that scientists *do not* care about is as important as identifying aspects that are really necessary.

Moreover, XML databases and XQuery are not yet in widespread use compared with relational databases, SQL, and other mature technologies. So our short-term goal is to understand what provenance is *in general* so that we can adapt to any particular set of scientific provenance needs. In doing so, we will develop a foundation for (XML or relational) database systems that provide rich forms of provenance tracking and can be configured to address the issues pertinent to a particular project.

5. Conclusions

The Web is revolutionizing the way science is performed. This revolution creates great opportunities, but also great risks. The Web’s integrity, reliability and reproducibility properties (or lack thereof) have a direct impact on the judgment of scientists and the quality of scientific results. We need to develop a *Scientific Web*, based on adapting Web technologies (including XML databases and other processing techniques) to provide a rigorous foundation for performing science online and retaining valuable results.

This paper has focused on provenance tracking for XML data. Although this is just one part of the overall problem, it is not yet well-understood. Our view is that just as numerical scientific computation ought to (and does) rest on solid mathematical foundations of real and complex analysis, differential equations and so on, so should symbolic scientific computation rest on the solid mathematical foundations of logic, programming language semantics and database theory. In particular, besides designing provenance-tracking systems according to ad hoc user specifications we should also seek out formal (and elegant) characterizations of provenance that explain why this information is useful and meaningful. The models we have developed in this paper, we hope, represent a first step towards this goal.

References

- [1] UAL shares fall as old story surfaces online. *Wall Street Journal*, September 9, 2008.
- [2] Special issue: The first provenance challenge. *Concurrency and Computation: Practice and Experience*, 20(5):409–418, 2008.

- [3] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *POPL*, pages 147–160. ACM Press, 1999.
- [4] Martín Abadi, Butler Lampson, and Jean-Jacques Lévy. Analysis and caching of dependencies. In *ICFP*, pages 83–91. ACM Press, 1996.
- [5] William A. Baumgartner, Jr., K. Bretonnel Cohen, Lynne M. Fox, George Acquaah-Mensah, and Lawrence Hunter. Manual curation is not sufficient for annotation of genomic databases. *Bioinformatics*, 23:i41–i48, 2007.
- [6] Catriel Beeri, Tova Milo, and Paula Ta-Shma. On genericity and parametricity (extended abstract). In *PODS 1996: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 104–116, New York, NY, USA, 1996. ACM Press.
- [7] Catriel Beeri, Tova Milo, and Paula Ta-Shma. Towards a language for the fully generic queries. In *DBPL-6: Proceedings of the 6th International Workshop on Database Programming Languages*, pages 239–259, London, UK, 1998. Springer-Verlag.
- [8] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *SIGMOD*, pages 539–550, 2006.
- [9] Peter Buneman, James Cheney, Wang-Chiew Tan, and Stijn Vansummeren. Curated databases. In *PODS*, pages 1–12, 2008.
- [10] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. *ACM Transactions on Database Systems*, 2008. Accepted for publication.
- [11] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and where: A characterization of data provenance. In *ICDT*, number 1973 in LNCS, pages 316–330. Springer, 2001.
- [12] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. On propagation of deletions and annotations through views. In *PODS*, pages 150–158, 2002.
- [13] Don Chamberlin, Daniela Florescu, Jim Melton, Jonathan Robie, and Jérôme Siméon. XQuery update facility. W3C Candidate Recommendation, March 2008. <http://www.w3c.org/TR/xquery-update-10/>.
- [14] James Cheney. LUX: A lightweight, statically typed XML update language. In *ACM SIGPLAN Workshop on Programming Language Technology and XML (PLAN-X 2007)*, pages 25–36, 2007.
- [15] James Cheney. FLUX: Functional Updates for XML. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 3–14, 2008.
- [16] James Cheney, Amal Ahmed, and Umut A. Acar. Provenance as dependency analysis. In *DBPL*, volume 4797 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2007.
- [17] Oracle Corporation. Oracle audit vault 10.2.3. <http://www.oracle.com/database/audit-vault.html>.
- [18] 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.
- [19] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1345–1350, New York, NY, USA, 2008. ACM.
- [20] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
- [21] Denise Draper, Peter Fankhauser, Mary Fernández, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, and Philip Wadler. XQuery 1.0 and XPath 2.0 formal semantics. W3C Recommendation, January 2007. <http://www.w3.org/TR/xquery-semantics/>.
- [22] J. Nathan Foster, Todd J. Green, and Val Tannen. Annotated XML: queries and provenance. In *PODS*, pages 271–280, 2008.
- [23] Michael Y. Galperin. The molecular biology database collection: 2008 update. *Nucleic Acids Research*, 36, 2008.
- [24] Floris Geerts and Antonella Poggi. On BP-complete query languages on K -relations. In *Workshop on Logic in Databases (LID)*, 2008. Available online: <http://conferenze.dei.polimi.it/lid2008/LID08geerts.pdf>.
- [25] Carole Goble and David de Roure. The Grid: An application of the Semantic Web. *SIGMOD Record*, 31(4):65–70, December 2002.
- [26] Todd J. Green. Containment of conjunctive queries on annotated relations. In *ICDT*, Saint Petersburg, Russia, March 2009 (to appear).
- [27] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40. ACM, 2007.
- [28] Erika Check Hayden. Protein prize up for grabs after retraction. *Nature News*, February 2008. doi:10.1038/news.2008.569.
- [29] Jan Hidders, Natalia Kwasnikowska, Jacek Sroka, Jerzy Tyszkiewicz, and Jan Van den Bussche. A formal model of dataflow repositories. In Sarah Cohen Boulakia and Val Tannen, editors, *DILS*, volume 4544 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 2007.
- [30] Limin Jia, Jeffrey A. Vaughan, Karl Mazurak, Jianzhou Zhao, Luke Zarko, Joseph Schorr, and Steve Zdancewic. Aura: a programming language for authorization and audit. *SIGPLAN Not.*, 43(9):27–38, 2008.
- [31] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.
- [32] Clifford A. Lynch. When documents deceive: trust and provenance as new factors for information retrieval in a tangled web. *J. Am. Soc. Inf. Sci. Technol.*, 52(1):12–17, 2001.
- [33] Greg Miller. A scientist’s nightmare: Software problem leads to five retractions. *Science*, 314(5807):1856–1857, December 2006.
- [34] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference*, pages 43–56. USENIX, June 2006.
- [35] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.
- [36] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, pages 1099–1110, New York, NY, USA, 2008. ACM.
- [37] John C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP Congress*, pages 513–523, 1983.
- [38] Andrei Sabelfeld and Andrew Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
- [39] Fred B. Schneider, J. Gregory Morrisett, and Robert Harper. A language-based approach to security. In Reinhard Wilhelm, editor, *Informatics*, volume 2000 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2001.
- [40] Sam Shah, Craig A. N. Soules, Gregory R. Ganger, and Brian D. Noble. Using provenance to aid in personal file search. In *USENIX Annual Technical Conference*, pages 171–184. USENIX, 2007.
- [41] Craig A. N. Soules, Garth R. Goodson, John D. Strunk, and Gregory R. Ganger. Metadata efficiency in versioning file systems. In *FAST*. USENIX, 2003.
- [42] John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A. N. Soules, and Gregory R. Ganger. Self-securing storage: Protecting data in compromised systems. In *OSDI*, pages 165–180, 2000.
- [43] Jeffrey A. Vaughan, Limin Jia, Karl Mazurak, and Steve Zdancewic. Evidence-based audit. In *CSF*, pages 177–191. IEEE, 2008.