Provenance as Dependency Analysis

James Cheney[1], Amal Ahmed[2], and Umut A. Acar[2] *

[1] University of Edinburgh
[2] Toyota Technological Institute at Chicago

Abstract. Provenance is information recording the source, derivation, or history of some information. Provenance tracking has been studied in a variety of settings; however, although many design points have been explored, the mathematical or semantic foundations of data provenance have received comparatively little attention. In this paper, we argue that dependency analysis techniques familiar from program analysis and program slicing provide a formal foundation for forms of provenance that are intended to show how (part of) the output of a query depends on (parts of) its input. We introduce a semantic characterization of such *dependency provenance*, show that this form of provenance is not computable, and provide dynamic and static approximation techniques.

1 Introduction

Provenance is information about the origin, ownership, influences upon, or other *historical* or *contextual* information about an object. Such information has many applications, including evaluating integrity or authenticity claims, establishing the chain of custody of or responsibility for an object, detecting and repairing errors, and memoization and caching of the results of computations. Provenance is particularly important in scientific computation and recordkeeping, since it is considered essential for ensuring the repeatability of experiments and judging the scientific value of their results.

Provenance tracking has been studied in a variety of settings, including databases, file systems, and scientific workflows; indeed, many familiar systems provide simple forms of provenance, such as the timestamp and ownership metadata in file systems, system logs, and version control systems. Although a wide variety of design points have been explored [8, 17, 21], there is relatively little understanding of the relationships among techniques or of the design considerations that should be taken into account when developing or evaluating an approach to provenance. The mathematical or semantic foundations of data provenance have received comparatively little attention, with a few relatively recent exceptions [10, 11, 15, 19].

Most prior approaches have invoked intuitive concepts such as *contribution*, *influence*, and *relevance* as motivation for their definitions of provenance. These intuitions seem adequate when considering monotone relational queries, but tend to break down when negation, grouping, or aggregation are considered. These intuitions have also motivated rigorous approaches to seemingly quite different problems, such as aiding debugging via *program slicing* [7, 16, 24], supporting efficient memoization and

* Cheney was supported by EPSRC grant R37476. Acar was supported by an Intel gift.

(a)

Protein			
ID	Name	MW	···
$\mathbf{p_1}$	thioredoxin	**11.8**	···
$\mathbf{p_2}$	flavodoxin	19.7	···
$\mathbf{p_3}$	ferredoxin	12.3	···
$\mathbf{p_4}$	ArgR	**−700**	···
$\mathbf{p_5}$	CheW	18.1	···
⋮	⋮	⋮	⋮

EnzReact	
PID	RID
$\mathbf{p_1}$	$\mathbf{r_1}$
$\mathbf{p_2}$	$\mathbf{r_1}$
$\mathbf{p_1}$	$\mathbf{r_2}$
$\mathbf{p_4}$	$\mathbf{r_2}$
$\mathbf{p_5}$	$\mathbf{r_3}$
⋮	⋮

Reaction		
ID	Name	···
$\mathbf{r_1}$	thia-phos + ATP = thi diphos + ADP	···
$\mathbf{r_2}$	H_2O + an acyl phos → phos + a carboxylate	···
$\mathbf{r_3}$	D-ribose-5-phosp = D-ribulose-5-phos	···
$\mathbf{r_4}$	β-D-gluc-6-phos = fruct-6-phos	···
$\mathbf{r_5}$	panteth 4′-phos + ATP = dephos-CoA + diphos	···
⋮	⋮	⋮

(b)

```
SELECT R.Name as Name, AVERAGE(P.MW) as AvgMW
  FROM Protein P, EnzymaticReaction ER, Reaction R
 WHERE P.ID = ER.ProteinID, ER.ReactionID = R.ID
GROUP BY R.Name
```

(c)

Name	AvgMW
thia-phos + ATP = thi diphos + ADP	15.75
H_2O + an acyl phos → phos + a carboxylate	*-338.2*
D-ribose-5-phosp = D-ribulose-5-phos	18.1
⋮	⋮

Fig. 1. Example (a) input, (b) query, and (b) output data; input field values relevant to *italicized* erroneous output value are highlighted in **bold**

caching [2, 4], and improving program security using information flow analysis [20]. As Abadi et al. have argued [1], slicing, information flow, and several other program analysis techniques can all be understood in terms of dependence.

In this paper, we argue that these *dependency analysis* and *slicing* techniques familiar from programming languages provide a suitable foundation for an interesting class of provenance techniques. To illustrate our approach, consider the (realistic, but biologically inaccurate) input data shown in Figure 1(a) and the query in Figure 1(b) which calculates the average molecular weights of proteins involved in each reaction. The result of this query is shown in Figure 1(c).

Since the MW field contains the molecular weight of a protein, it is clearly an error for the italicized value in the result to be negative. To track down the source of the error, it would be helpful to know which parts of the input *contributed to*, or were *relevant to*, the erroneous part of the output. We can formalize this intuition by saying that a part of the output *depends on* a part of the input if a change to the input part may result in a change to the output part. This is analogous to program slicing [24], a debugging aid that identifies the parts of a *program* on which a program output depends.

In this example, the input field values that the erroneous output AvgMW-value depends on are highlighted in bold. The dependences include the two summed MW values

and the ID fields which are compared by the selection and grouping query. These ID fields must be included because a change to any one of them could result in a change to the italicized output value—for example, changing the occurrence of p_4 in table EnzymaticReaction. On the other hand, the names of the proteins and reactions are *irrelevant* to the output AvgMW.

This example is simplistic, but the ability to concisely explain which parts of the input influence each part of the output is much more important if we consider a realistic database with perhaps tens or hundreds of columns per table and thousands or millions of rows. Moreover, dependence information can also be useful for a variety of other applications, including estimating the *quality* or *freshness* of data in a query result by aggregating timestamps or quality annotations on the relevant inputs.

In this paper, we argue that data dependence provides a solid semantic foundation for generalizing why-provenance, lineage, and similar techniques. We consider the full nested relational calculus with set difference, equality, grouping and aggregation operations, and functions on basic types. We consider annotation-propagating semantics for such queries and define a property called *dependency-correctness*, which, intuitively, means that the annotations produced by a query correctly show all parts of the output that may change if a part of the input is changed.

The structure of the rest of this paper is as follows. We briefly review the nested relational calculus in Section 2. We then introduce (in Section 3) the annotation-propagation model and define dependency-correctness. In Section 3.1 we describe a dynamic *provenance-tracking* semantics that is dependency-correct. We also (Section 3.2) introduce a type-based *provenance analysis* which is less accurate than provenance tracking, but can be performed statically; we also prove its correctness relative to dynamic provenance tracking. We discuss our results and a preliminary implementation in Section 3.3 and discuss related and future work and conclude in Sections 4–5. Full proofs of our main results are provided in a companion technical report [14].

2 Background

We assume some familiarity with the *nested relational calculus* (NRC) [13], which is closely related to *monad algebra* [22]. We consider multiset (or bag) collections rather than sets. The types and expressions of our variant of NRC are as follows:

$$\tau ::= \mathsf{bool} \mid \mathsf{int} \mid \tau_1 \times \tau_2 \mid \{\tau\}$$
$$e ::= x \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid (e_1, e_2) \mid \pi_i(e) \mid b \mid i \mid \neg e \mid e_1 \wedge e_2 \mid e_1 + e_2 \mid \mathsf{sum}(e)$$
$$\mid\ e_1 \approx e_2 \mid \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid \emptyset \mid \{e\} \mid e_1 \cup e_2 \mid e_1 - e_2 \mid \{e_2 \mid x \in e_1\} \mid \bigcup e$$

Here, $i \in \mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$ represents integer constants and $b \in \mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ denotes Boolean constants. The type $\{\tau\}$ describes *collections* of elements of type τ; in this paper, we consider collections to be *bags* (multisets). The bag operations include $\emptyset$, the constant empty bag; singletons $\{e\}$; bag union and difference; comprehension $\{e_2 \mid x \in e_1\}$; and flattening $\bigcup e$. By convention, we write $\{e_1, \ldots, e_n\}$ as syntactic sugar for $\{e_1\} \cup \cdots \cup \{e_n\}$. Finally, we include sum, a typical aggregation operation, which adds together all of the elements of a bag and produces a value; e.g.

$\mathsf{sum}\{1,2,3\} = 6$ (by convention, $\mathsf{sum}(\emptyset) = 0$). We syntactically distinguish between NRC's equality operation $\approx$ and mathematical equality $=$.

Types Query language expressions can be typechecked using standard techniques. Contexts Γ are lists of pairs of variables and types $x_1 : \tau_1, \ldots, x_n : \tau_n$, where $x_1, \ldots, x_n$ are distinct. The rules for typechecking expressions are shown in Figure 2.

Semantics We write $\mathcal{M}_{\mathsf{fin}}(X)$ for the set of all finite bags with elements drawn from X. The (standard) interpretation of base types as sets of values is as follows:

$$\mathcal{T}[\![\mathsf{bool}]\!] = \mathbb{B} = \{\mathsf{true}, \mathsf{false}\} \qquad \mathcal{T}[\![\tau_1 \times \tau_2]\!] = \mathcal{T}[\![\tau_1]\!] \times \mathcal{T}[\![\tau_2]\!]$$
$$\mathcal{T}[\![\mathsf{int}]\!] = \mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\} \qquad \mathcal{T}[\![\{\tau\}]\!] = \mathcal{M}_{\mathsf{fin}}(\mathcal{T}[\![\tau]\!])$$

An environment γ is a function from variables to values. We define the set of environments matching context Γ as $\mathcal{T}[\![\Gamma]\!] = \{\gamma \mid \forall x \in dom(\Gamma).\gamma(x) \in \mathcal{T}[\![\Gamma(x)]\!]\}$.

Figure 3 gives the semantics of queries. Note that we overload notation for pair projection π_i and bag operations such as $\cup$ and $\bigcup$; also, if S is a bag of integers, then $\sum S$ is the sum of their values (taking $\sum \emptyset = 0$). It is straightforward to show that

Lemma 1. *If $\Gamma \vdash e : \tau$ then $\mathcal{E}[\![e]\!] : \mathcal{T}[\![\Gamma]\!] \to \mathcal{T}[\![\tau]\!]$.*

As discussed in previous work [13], the NRC can express a wide variety of queries including ordinary relational queries as well as grouping and aggregation. We do not consider incomplete information (`NULL` values). Additional primitive functions and relations such as average can also be added without difficulty (in fact, average is definable in our language). For example, using more readable named record, comprehensions, and patterns, the SQL query from the Figure 1(b) can be defined as

$\mathsf{let}\ X = \{(r.Name, p.MW) \mid r \in R, er \in ER, p \in P, er.RID = r.ID, p.ID = er.PID\}\ \mathsf{in}$
$\{(n, \mathsf{average}\{mw \mid (n', mw) \in X, n = n'\}) \mid (n, \_) \in X\}$

Additional examples are shown in Figure 4.

3 Annotations, Provenance and Dependence

We wish to define *dependency provenance* as information relating each part of the output of a query to a set of parts of the input on which the output part depends. Collection types such as sets and bags are unordered and lack a natural way to "address" parts of values, so we must introduce one. One technique (familiar from many program analyses as well as other work on provenance [10, 23]) is to enrich the data model with *annotations* that can be used to refer to parts of the value. We can then infer provenance information from functions on annotated values by observing how such functions propagate annotations; conversely, we can define provenance-tracking semantics by enriching ordinary functions with annotation-propagation behavior. In this section, we show how to define dependency provenance in this way. In the next sections, we will show how to compute dynamic and static dependency provenance for NRC queries using annotations.

We define *annotated values (a-values)* v, *raw values (r-values)* w, and *multisets of annotated values* V as follows:

$$v ::= w^{\Phi} \qquad w ::= i \mid b \mid (v_1, v_2) \mid V \qquad V ::= \{v_1, \ldots, v_n\}$$

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{x{:}\tau \in \Gamma}{\Gamma \vdash x : \tau} \quad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x{:}\tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2} \quad \frac{i \in \mathbb{Z}}{\Gamma \vdash i : \mathsf{int}} \quad \frac{\Gamma \vdash e_1 : \mathsf{int} \quad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}$$

$$\frac{\Gamma \vdash e : \{\mathsf{int}\}}{\Gamma \vdash \mathsf{sum}(e) : \mathsf{int}} \quad \frac{b \in \mathbb{B}}{\Gamma \vdash b : \mathsf{bool}} \quad \frac{\Gamma \vdash e_0 : \mathsf{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau} \quad \frac{\Gamma \vdash e : \mathsf{bool}}{\Gamma \vdash \neg e : \mathsf{bool}}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : \mathsf{bool}}{\Gamma \vdash e_1 \wedge e_2 : \mathsf{bool}} \quad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \quad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \pi_i(e) : \tau_i}$$

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 \approx e_2 : \mathsf{bool}} \quad \frac{}{\Gamma \vdash \emptyset : \{\tau\}} \quad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \{e\} : \{\tau\}} \quad \frac{\Gamma \vdash e_1 : \{\tau\} \quad \Gamma \vdash e_2 : \{\tau\}}{\Gamma \vdash e_1 \cup e_2 : \{\tau\}}$$

$$\frac{\Gamma \vdash e_1 : \{\tau\} \quad \Gamma \vdash e_2 : \{\tau\}}{\Gamma \vdash e_1 - e_2 : \{\tau\}} \quad \frac{\Gamma \vdash e_1 : \{\tau_1\} \quad \Gamma, x{:}\tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \{e_2 \mid x \in e_1\} : \{\tau_2\}} \quad \frac{\Gamma \vdash e : \{\{\tau\}\}}{\Gamma \vdash \bigcup e : \{\tau\}}$$

Fig. 2. Well-formed query expressions

$$\begin{aligned}
\mathcal{E}[\![x]\!]\gamma &= \gamma(x) & \mathcal{E}[\![\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2]\!]\gamma &= \mathcal{E}[\![e_2]\!]\gamma[x \mapsto \mathcal{E}[\![e_1]\!]\gamma] \\
\mathcal{E}[\![i]\!]\gamma &= i & \mathcal{E}[\![e_1 + e_2]\!]\gamma &= \mathcal{E}[\![e_1]\!]\gamma + \mathcal{E}[\![e_2]\!]\gamma \\
\mathcal{E}[\![\mathsf{sum}(e)]\!]\gamma &= \textstyle\sum \mathcal{E}[\![e]\!]\gamma & \mathcal{E}[\![b]\!]\gamma &= b \\
\mathcal{E}[\![\neg e]\!]\gamma &= \neg \mathcal{E}[\![e]\!]\gamma & \mathcal{E}[\![e_1 \wedge e_2]\!]\gamma &= \mathcal{E}[\![e_1]\!]\gamma \wedge \mathcal{E}[\![e_2]\!]\gamma \\
\mathcal{E}[\![(e_1, e_2)]\!]\gamma &= (\mathcal{E}[\![e_1]\!]\gamma, \mathcal{E}[\![e_2]\!]\gamma) & \mathcal{E}[\![\pi_i(e)]\!]\gamma &= \pi_i(\mathcal{E}[\![e]\!]\gamma) \\
\mathcal{E}[\![\emptyset]\!]\gamma &= \emptyset & \mathcal{E}[\![\{e\}]\!]\gamma &= \{\mathcal{E}[\![e]\!]\gamma\} \\
\mathcal{E}[\![e_1 \cup e_2]\!]\gamma &= \mathcal{E}[\![e_1]\!]\gamma \cup \mathcal{E}[\![e_2]\!]\gamma & \mathcal{E}[\![e_1 - e_2]\!]\gamma &= \mathcal{E}[\![e_1]\!]\gamma - \mathcal{E}[\![e_2]\!]\gamma \\
\mathcal{E}[\![\textstyle\bigcup e]\!]\gamma &= \textstyle\bigcup \mathcal{E}[\![e]\!]\gamma & \mathcal{E}[\![\{e \mid x \in e_0\}]\!]\gamma &= \{\mathcal{E}[\![e]\!]\gamma[x \mapsto v] \mid v \in \mathcal{E}[\![e_0]\!]\gamma\}
\end{aligned}$$

$$\mathcal{E}[\![\mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]\!]\gamma = \begin{cases} \mathcal{E}[\![e_1]\!]\gamma \text{ if } \mathcal{E}[\![e_0]\!]\gamma = \mathsf{true} \\ \mathcal{E}[\![e_2]\!]\gamma \text{ if } \mathcal{E}[\![e_0]\!]\gamma = \mathsf{false} \end{cases}$$

$$\mathcal{E}[\![e_1 \approx e_2]\!]\gamma = \begin{cases} \mathsf{true} \text{ if } \mathcal{E}[\![e_1]\!]\gamma = \mathcal{E}[\![e_2]\!]\gamma \\ \mathsf{false} \text{ if } \mathcal{E}[\![e_1]\!]\gamma \neq \mathcal{E}[\![e_2]\!]\gamma \end{cases}$$

Fig. 3. Semantics of query expressions

$$\begin{aligned}
\Pi_A(R) &= \{x.A \mid x \in R\} \\
\sigma_{A=B}(R) &= \textstyle\bigcup \{\mathsf{if}\ x.A = x.B\ \mathsf{then}\ \{x\}\ \mathsf{else}\ \emptyset \mid x \in R\} \\
R \times S &= \{(A : x.A, B : x.B, C : y.C, D : y.D, E : y.E) \mid x \in R, y \in S\} \\
\Pi_{BE}(\sigma_{A=D}(R \times S)) &= \{\mathsf{if}\ x.A = y.D\ \mathsf{then}\ \{(B : x.B, E : y.E)\}\ \mathsf{else}\ \emptyset \mid x \in R, y \in S\} \\
R \cup \rho_{A/C,B/D}(\Pi_{CD}(S)) &= R \cup \{(A : y, C, B : y.D) \mid y \in S\} \\
R - \rho_{A/D,B/E}(\Pi_{DE}(S)) &= R - \{(A : y, D, B : y.E) \mid y \in S\} \\
\mathsf{sum}(\Pi_A(R)) &= \mathsf{sum}\{x.A \mid x \in R\} \\
\mathsf{count}(R) &= \mathsf{sum}\{1 \mid x \in R\}
\end{aligned}$$

Fig. 4. Example queries

For us, annotations are *sets* $\Phi \subseteq \mathsf{Color}$ of values from some atomic data type Color of *colors*. We often omit set brackets in the annotations, for example writing $w^{a,b,c}$ instead of $w^{\{a,b,c\}}$ and w instead of $w^{\emptyset}$. An a-value v is said to be *distinctly colored* if every part of it is colored with a singleton set $\{a\}$ and no color c is repeated anywhere in v.

For each type τ, we define the set $\mathcal{A}_0[\![\tau]\!]$ of annotated values of type τ as follows:

$$\mathcal{A}_0[\![\mathsf{bool}]\!] = \{b^{\Phi} \mid b \in \mathbb{B}\} \quad \mathcal{A}_0[\![\tau_1 \times \tau_2]\!] = \{(v_1, v_2)^{\Phi} \mid v_1 \in \mathcal{A}_0[\![\tau_1]\!], v_2 \in \mathcal{A}_0[\![\tau_2]\!]\}$$
$$\mathcal{A}_0[\![\mathsf{int}]\!] = \{i^{\Phi} \mid i \in \mathbb{Z}\} \qquad \mathcal{A}_0[\![\{\tau\}]\!] = \{V^{\Phi} \mid \forall v \in V. v \in \mathcal{A}_0[\![\tau]\!]\}$$

Annotated environments $\widehat{\gamma}$ map variables to annotated values. We define the set of annotated environments matching context Γ as $\mathcal{A}_0[\![\Gamma]\!] = \{\widehat{\gamma} \mid \forall x \in dom(\Gamma). \widehat{\gamma}(x) \in \mathcal{A}_0[\![\Gamma(x)]\!]\}$.

We define an erasure function $|-|$, mapping a-values to ordinary values (and, abusing notation, also mapping r-values to ordinary values), as follows:

$$|i| = i \quad |b| = b \quad |(v_1, v_2)| = (|v_1|, |v_2|) \quad |\{V\}| = \{|v| \mid v \in V\} \quad |w^{\Phi}| = |w|$$

and an annotation extraction function $\|-\|$ which extracts the set of all colors mentioned anywhere in an a-value or r-value, defined by taking $\|w^{\Phi}\| = \Phi \cup \|w\|$ and

$$\|i\| = \emptyset \quad \|b\| = \emptyset \quad \|(v_1, v_2)\| = \|v_1\| \cup \|v_2\| \quad \|\{V\}\| = \bigcup \{\|v\| \mid v \in V\}$$

Two a-values are said to be *compatible* (written $v \cong v'$) if $|v| = |v'|$.

We now consider annotated functions (a-functions) $F : \mathcal{A}_0[\![\Gamma]\!] \to \mathcal{A}_0[\![\tau]\!]$ on a-values. Recall that we plan to define provenance for functions $f : \mathcal{T}[\![\Gamma]\!] \to \mathcal{T}[\![\tau]\!]$ by observing how a-functions transform annotations. For this to make sense, we first need to restrict attention to a-functions F whose behavior is consistent with that of some function f; that is, such that $\forall v \in \mathcal{A}_0[\![\Gamma]\!]. f(|v|) = |F(v)|$. If this is the case, then we say that the a-function F is an enrichment of f; there can be at most one such f, so we sometimes write $|F|$ for f. Of course, many a-functions are not enrichments of any ordinary function: for example, suppose $F_0(1^a) = 1^a$ while $F_0(1^b) = 2^b$. It may be of interest to semantically characterize the a-functions that are enrichments of ordinary functions, by analogy with generic queries in relational databases and color-invariance in [10]; while this would be important for studying expressiveness, in this paper we simply restrict attention to a-functions F that are enrichments of ordinary functions, that is, for which $|F|$ exists.

Dependency-correctness Intuitively, an a-function F is dependency-correct if its output annotations tell us how changes to parts of the input may affect parts of the output. First, we need to capture the intuitive notion of changing a value at a particular location:

Definition 1 (Equal except at c). *Two a-values v_1, v_2 are* equal except at c *($v_1 \equiv_c v_2$) provided that they have the same structure except possibly at subterms labeled with c; this relation is defined as follows:*

$$\frac{d \in \mathbb{B} \cup \mathbb{Z}}{d \equiv_c d} \quad \frac{v_1 \equiv_c v_1' \quad v_2 \equiv_c v_2'}{(v_1, v_2) \equiv_c (v_1', v_2')} \quad \frac{v_1 \equiv_c v_1' \quad \cdots \quad v_n \equiv_c v_n'}{\{v_1, \ldots, v_n\} \equiv_c \{v_1', \ldots, v_n'\}} \quad \frac{w_1 \equiv_c w_2}{w_1^{\Phi} \equiv_c w_2^{\Phi}} \quad \frac{c \in \Phi_1 \cap \Phi_2}{w_1^{\Phi_1} \equiv_c w_2^{\Phi_2}}$$

Example 1. Consider the two a-environments:

$$\widehat{\gamma} = (\mathrm{R} : \{(1^{c_1}, 3^{c_2}, 5^{c_3})^{b_1}, \ldots\}^a, \mathrm{S} : \cdots)$$
$$\widehat{\gamma}' = (\mathrm{R} : \{(2^{c_1}, 3^{c_2}, 5^{c_3})^{b_1}, \ldots\}^a, \mathrm{S} : \cdots)$$

We have $\widehat{\gamma} \equiv_a \widehat{\gamma}'$, $\widehat{\gamma} \equiv_{b_1} \widehat{\gamma}'$, and $\widehat{\gamma} \equiv_{c_1} \widehat{\gamma}'$, assuming that the elided portions are identical. For distinctly-colored values, a color serves as an address uniquely identifying a subterm. Thus, $\equiv_c$ relates a distinctly-colored value to a value which can be obtained by modifying the subterm "at c"; that is, if we write v_1 as $C[v_1']$ where C is a context and v_1' is the subterm labeled with c in v_1, and $v_1 \equiv_c v_2$, then $v_2 = C[v_2']$ for some subterm v_2' labeled with c. Note that v_2' and v_2 need not be distinctly colored, and that $\equiv_c$ makes sense for arbitrary a-values, not just distinctly colored ones.

Definition 2 (Dependency-correctness). *An a-function $F : \mathcal{A}_0[\![\Gamma]\!] \to \mathcal{A}_0[\![\tau]\!]$ is* dependency-correct *if for any $c \in \mathsf{Color}$ and $\widehat{\gamma}, \widehat{\gamma}' \in \mathcal{A}_0[\![\Gamma]\!]$ satisfying $\widehat{\gamma} \equiv_c \widehat{\gamma}'$, we have $F(\widehat{\gamma}) \equiv_c F(\widehat{\gamma}')$.*

Example 2. Recall $\widehat{\gamma}, \widehat{\gamma}'$ as in the previous example. Suppose

$$F(\widehat{\gamma}) = \{(1^{c_1}, 3^{c_2}, 5^{c_3})^{b_1}\}^a \ .$$

Since $\widehat{\gamma} \equiv_{c_1} \widehat{\gamma}'$, we know that $F(\widehat{\gamma}) \equiv_{c_1} F(\widehat{\gamma}')$ so we can see that $F(\widehat{\gamma}')$ must be of the form

$$\{(s^{c_1}, 3^{c_2}, 5^{c_3})^{b_1}\}^a$$

for some $n \in \mathbb{Z}$. We do *not* necessarily know that n must be 2; this is not captured by dependency-correctness.

More generally, dependency-correctness tells us that for any c, we must have $F(\widehat{\gamma}) = C[v_1, \ldots, v_n]$ and $F(\widehat{\gamma}') = C[v_1', \ldots, v_n']$, where $C[-, \ldots, -]$ is a context not mentioning c and $v_1, \ldots, v_n, v_1', \ldots, v_n'$ are labeled with c. Thus, F's annotations tell us which parts of the output (i.e., $v_1, \ldots, v_n$) *may* change if the input is changed at c. Dually, they also tell us what part of the output (i.e., $C[-, \ldots, -]$) *cannot* be changed by changing the input at c.

Of course, dependency-correctness does not uniquely characterize the annotation behavior of a given F. It is possible for the annotations to be dependency-correct but *inaccurate*. For example we can always trivially annotate each part of the output with *every* color appearing in the input. This, of course, tells us nothing about the function's behavior. In general, the fewer the annotations present in the output of a dependency-correct F, the more they tell us about F's behavior. We therefore consider a function F to be *minimally annotated* if no annotations can be removed from F's output for any v without damaging correctness.

We say that a query e is *constant* if $[\![e]\!]\gamma = v$ for some v and every suitable γ.

Proposition 1. *It is undecidable whether a Boolean NRC query is constant.*

Proof. Recall that query equivalence is undecidable for the (nested) relational calculus [3]; this holds even for queries $e(x), e'(x)$ over a single variable x. Given two such queries, consider the expression $\widehat{e} = e(x) \approx e'(x) \vee y$ (definable as $\neg(\neg(e(x) \approx e'(x)) \wedge \neg y)$). The result of this expression cannot be false everywhere since the disjunction is true for $y = \mathsf{true}$, so is $\widehat{e}$ is constant iff $[\![\widehat{e}]\!]v = \mathsf{true}$ for every v iff $e \equiv e'$.

$$
\begin{array}{rlrl}
(w^{\Phi})^{+\Phi_0} &= w^{\Phi\cup\Phi_0} & (i_1^{\Phi_1}) \mathbin{\widehat{+}} (i_2^{\Phi_2}) &= (i_1 + i_2)^{\Phi_1\cup\Phi_2} \\
\widehat{\neg}(b^{\Phi}) &= (\neg b)^{\Phi} & (b_1^{\Phi_1}) \mathbin{\widehat{\wedge}} (b_2^{\Phi_2}) &= (b_1 \wedge b_2)^{\Phi_1\cup\Phi_2} \\
\widehat{\pi}_i((v_1, v_2)^{\Phi}) &= v_i^{+\Phi} & (w_1^{\Phi_1}) \mathbin{\widehat{\cup}} (w_2^{\Phi_2}) &= (w_1 \cup w_2)^{\Phi_1\cup\Phi_2} \\
\widehat{\mathsf{cond}}(\mathsf{true}^{\Phi}, v_1, v_2) &= v_1^{+\Phi} & \widehat{\mathsf{cond}}(\mathsf{false}^{\Phi}, v_1, v_2) &= v_2^{+\Phi}
\end{array}
$$

$$
\begin{aligned}
\widehat{\sum}(\{v_1, \ldots, v_n\}^{\Phi}) &= (v_1 \mathbin{\widehat{+}} \cdots \mathbin{\widehat{+}} v_n)^{+\Phi} \\
\widehat{\bigcup}\{v_1, \ldots, v_n\}^{\Phi} &= (v_1 \mathbin{\widehat{\cup}} \cdots \mathbin{\widehat{\cup}} v_n)^{+\Phi} \\
\{v(x) \mid x \mathbin{\widehat{\in}} w^{\Phi}\} &= \{v(x) \mid x \in w\}^{\Phi} \\
(w_1^{\Phi_1}) \mathbin{\widehat{-}} (w_2^{\Phi_2}) &= \{v \in w_1 \mid |v| \notin |w_2|\}^{\Phi_1\cup\|w_1\|\cup\Phi_2\cup\|w_2\|} \\
v_1 \mathbin{\widehat{\approx}} v_2 &= \begin{cases} \mathsf{true}^{\|v_1\|\cup\|v_2\|} & |v_1| = |v_2| \\ \mathsf{false}^{\|v_1\|\cup\|v_2\|} & |v_1| \neq |v_2| \end{cases}
\end{aligned}
$$

Fig. 5. Auxiliary annotation-propagating operations

$$
\begin{array}{rlrl}
\mathcal{P}[\![x]\!]\widehat{\gamma} &= \widehat{\gamma}(x) & \mathcal{P}[\![\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2]\!]\widehat{\gamma} &= \mathcal{P}[\![e_2]\!]\widehat{\gamma}[x \mapsto \mathcal{P}[\![e_1]\!]\widehat{\gamma}] \\
\mathcal{P}[\![i]\!]\widehat{\gamma} &= i^{\emptyset} & \mathcal{P}[\![e_1 + e_2]\!]\widehat{\gamma} &= \mathcal{P}[\![e_1]\!]\widehat{\gamma} \mathbin{\widehat{+}} \mathcal{P}[\![e_2]\!]\widehat{\gamma} \\
\mathcal{P}[\![\mathsf{sum}(e)]\!]\widehat{\gamma} &= \widehat{\sum}\mathcal{P}[\![e]\!]\widehat{\gamma} & \mathcal{P}[\![b]\!]\widehat{\gamma} &= b^{\emptyset} \\
\mathcal{P}[\![\neg e]\!]\widehat{\gamma} &= \widehat{\neg}\mathcal{P}[\![e]\!]\widehat{\gamma} & \mathcal{P}[\![e_1 \wedge e_2]\!]\widehat{\gamma} &= \mathcal{P}[\![e_1]\!]\widehat{\gamma} \mathbin{\widehat{\wedge}} \mathcal{P}[\![e_2]\!]\widehat{\gamma} \\
\mathcal{P}[\![(e_1, e_2)]\!]\widehat{\gamma} &= (\mathcal{P}[\![e_1]\!]\widehat{\gamma}, \mathcal{P}[\![e_2]\!]\widehat{\gamma})^{\emptyset} & \mathcal{P}[\![\pi_i(e)]\!]\widehat{\gamma} &= \widehat{\pi}_i(\mathcal{P}[\![e]\!]\widehat{\gamma}) \\
\mathcal{P}[\![\emptyset]\!]\widehat{\gamma} &= \emptyset^{\emptyset} & \mathcal{P}[\![\{e\}]\!]\widehat{\gamma} &= \{\mathcal{P}[\![e]\!]\widehat{\gamma}\}^{\emptyset} \\
\mathcal{P}[\![e_1 \cup e_2]\!]\widehat{\gamma} &= \mathcal{P}[\![e_1]\!]\widehat{\gamma} \mathbin{\widehat{\cup}} \mathcal{P}[\![e_2]\!]\widehat{\gamma} & \mathcal{P}[\![e_1 - e_2]\!]\widehat{\gamma} &= \mathcal{P}[\![e_1]\!]\widehat{\gamma} \mathbin{\widehat{-}} \mathcal{P}[\![e_2]\!]\widehat{\gamma} \\
\mathcal{P}[\![\bigcup e]\!]\widehat{\gamma} &= \widehat{\bigcup} \mathcal{P}[\![e]\!]\widehat{\gamma} & \mathcal{P}[\![\{e \mid x \in e_0\}]\!]\widehat{\gamma} &= \{\mathcal{P}[\![e]\!]\widehat{\gamma}[x \mapsto v] \mid v \mathbin{\widehat{\in}} \mathcal{P}[\![e_0]\!]\widehat{\gamma})\} \\
\mathcal{P}[\![e_1 \approx e_2]\!]\widehat{\gamma} &= \mathcal{P}[\![e_1]\!]\widehat{\gamma} \mathbin{\widehat{\approx}} \mathcal{P}[\![e_2]\!]\widehat{\gamma} & \mathcal{P}[\![\mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]\!]\widehat{\gamma} &= \widehat{\mathsf{cond}}(\mathcal{P}[\![e_0]\!]\widehat{\gamma}, \mathcal{P}[\![e_1]\!]\widehat{\gamma}, \mathcal{P}[\![e_2]\!]\widehat{\gamma})
\end{array}
$$

Fig. 6. Provenance-tracking semantics

Clearly, an annotation is needed on the result of a Boolean query if and only if the query is not a constant, so finding minimal annotations is undecidable. As a result, we cannot expect to be able to compute dependency-correct annotations with perfect accuracy. It is important to note, though, that dependency-tracking is hard even if we leave out the $e_1 - e_2$ or $e_1 \approx e_2$ operators. For example, we can reduce (coNP-hard) propositional validity problems to dependency-tracking for ordinary Boolean expressions or conditionals.

3.1 Dynamic Provenance Tracking

We now consider a *provenance tracking* approach in which we interpret each expressions e as dependency-correct a-functions $\mathcal{P}[\![e]\!]$. The definition of the provenance-tracking semantics is shown in Figure 6. Auxiliary operations are used to define $\mathcal{P}[\![-]\!]$; these are shown in Figure 5. In particular, note that we define $(w^{\Phi})^{+\Psi} = w^{\Phi\cup\Psi}$.

Many cases involving ordinary programming constructs are self-explanatory. Constants always have empty annotations: nothing in the input can affect them. Built-in

functions such as $+, \wedge, \neg$ propagate all annotations on their arguments to the result. For a conditional if e_0 then e_1 else e_2, the result is the result of evaluating e_1 or e_2, combined with the annotations of e_0. A constructed pair has an empty top-level annotation; in a projection, the top-level annotation of the pair is merged with that of the returned value.

The cases involving collection types deserve some explanation. The empty set is a constant, so has an empty top-level annotation. Similarly, a singleton set constructor has an empty annotation. For union, we take the union of the underlying bags (of annotated values) and fuse the top-level annotations. For comprehension, we leave the top-level annotation alone. For flattening $\bigcup e$, we take the lifted union ($\widehat{\cup}$) of the elements of e and add the top-level annotation of e. Similarly, $\widehat{\mathsf{sum}}(e)$ uses $\widehat{+}$ to add together the elements of e, fusing their annotations with that of e. For set difference, to ensure dependency correctness, we must conservatively include all of the colors present on either side in the annotation of the top-level expression. Similarly, for equality tests, we must include all of the colors present in either value in the result annotation.

Remark 1. Our approach to handling negation and equality is somewhat awkward since it may result in very large annotations. For example, consider $\{1^a, 2^b\}^c - \{1^d, 3^e\}^f$: clearly, changing any of the input locations a, b, c, d, e, f can cause the output to change. In contrast, other approaches such as lineage associate tuple $t \in R - S$ only with $t \in R$ and all tuples of S. This may seem more "accurate", but it is not dependency-correct. On the other hand, our approach can also be more "accurate" than lineage in the presence of negation; for example, in $\{1\} - \{\pi_1(x) \mid x \in S\}$, our approach will indicate that the output does not depend on the second components of elements of S, whereas the lineage of this query include all the records in S.

However, although our approach to negation and equality has pathological behavior in some cases, it does seem to provide useful provenance for typical queries. Developing more sophisticated forms of dependence that are better-behaved in the presence of negation or equality is an interesting area for future work.

Example 3. Consider an annotated input environment $\widehat{\gamma}$, shown in Figure 7(a), of schema $R : (A : \mathsf{int}, B : \mathsf{int}), S(C : \mathsf{int}, D : \mathsf{int}, E : \mathsf{int})$ (we use more compact relational schema notation with field names for readability). Figure 7(b) shows the provenance tracking semantics of the example queries from Figure 4. We write a_{123} as an abbreviation for a_1, a_2, a_3, etc. Note that in the count example query, the output depends on *no* individual of the input; we cannot change the number of elements of a multiset by changing field values. However, in a query such as $\mathsf{count}(\sigma_{A=B}(R))$, the result depends on all of the A and B fields.

These annotated results can easily be used to "highlight" the parts of the input that may be relevant to a part of the output, by examining the annotations appearing above the output part of interest. This is how the example in Figure 1 was constructed.

Note that equivalent expressions $e \equiv e'$ need not satisfy $\mathcal{P}[\![e]\!] \equiv \mathcal{P}[\![e']\!]$; for example, $x - x \equiv \emptyset$ but $\mathcal{P}[\![x - x]\!] \not\equiv \emptyset^{\emptyset}$.

We summarize the main results concerning $\mathcal{P}[\![-]\!]$ as follows:

Theorem 1. *If $\Gamma \vdash e : \tau$ then (1) $\mathcal{P}[\![e]\!] : \mathcal{A}_0[\![\Gamma]\!] \to \mathcal{A}_0[\![\tau]\!]$, (2) $|\mathcal{P}[\![e]\!]| = \mathcal{E}[\![e]\!]$, and (3) $\mathcal{P}[\![e]\!]$ is dependency-correct.*

(a)

$$\widehat{\gamma} = [R := \{(A : 1^{a_1}, B : 1^{b_1}), (A : 1^{a_2}, B : 2^{b_2}), (A : 2^{a_3}, B : 3^{b_3})\},$$
$$S := \{(C : 1^{c_1}, D : 2^{d_1}, E : 3^{e_1}), (C : 1^{c_2}, D : 1^{d_2}, E : 4^{e_2})\}]$$

(b)

$$\begin{aligned}
\mathcal{P}[\![\Pi_A(R)]\!]\widehat{\gamma} &= \{(A : 1^{a_1}), (A : 1^{a_2}), (A : 2^{a_3})\} \\
\mathcal{P}[\![\sigma_{A=B}(R)]\!]\widehat{\gamma} &= \{(A : 1^{a_1}, B : 1^{b_1})\}^{a_1,b_1} \\
\mathcal{P}[\![R \times S]\!]\widehat{\gamma} &= \{(A : 1^{a_1}, B : 1^{b_1}, C : 1^{c_1}, D : 2^{d_1}, E : 3^{e_1}), \\
&\quad (A : 1^{a_1}, B : 1^{b_1}, C : 1^{c_2}, D : 1^{d_2}, E : 4^{e_2}), \ldots\} \\
\mathcal{P}[\![\Pi_{BE}(\sigma_{A=D}(R \times S))]\!]\widehat{\gamma} &= \{(B : 1^{b_1}, E : 4^{e_2}), (B : 2^{b_2}, E : 4^{e_2}), \\
&\quad (B : 3^{b_3}, E : 3^{e_1})\}^{a_{123},d_{12}} \\
\mathcal{P}[\![R \cup \rho_{A/C,B/D}(\Pi_{CD}(S))]\!]\widehat{\gamma} &= \{(A : 1^{a_1}, B : 1^{b_1}), (A : 1^{a_2}, B : 2^{b_2}), (A : 2^{a_3}, B : 3^{b_3}), \\
&\quad (A : 1^{c_1}, B : 2^{d_1}), (A : 1^{c_2}, B : 1^{d_2})\} \\
\mathcal{P}[\![R - \rho_{A/D,B/E}(\Pi_{DE}(S))]\!]\widehat{\gamma} &= \{(A : 1^{a_1}, B : 1^{b_2}), (A : 1^{a_2}, B : 2^{b_2})\}^{a_{123},b_{123},d_{12},e_{12}} \\
\mathcal{P}[\![\mathsf{sum}(\Pi_A(R))]\!]\widehat{\gamma} &= 4^{a_1,a_2,a_3} \qquad \mathcal{P}[\![\mathsf{count}(R)]\!]\widehat{\gamma} = 3
\end{aligned}$$

Fig. 7. (a) Annotated input environment (b) Examples of provenance tracking

3.2 Static Provenance Analysis

Although it can often be quite accurate, dynamic provenance seems expensive to compute and nontrivial to implement in a standard relational database system. Moreover, dynamic analysis cannot tell us anything about a query without looking at (annotated) input data. In this section we consider a *static provenance analysis* which statically approximates the dynamic provenance, but can be calculated more easily and without accessing the input.

We formulate the analysis as a type-based analysis; annotated types (a-types) $\widehat{\tau}$ and raw types (r-types) ω are defined as follows:

$$\widehat{\tau} ::= \omega^{\Phi} \qquad \omega ::= \mathsf{int} \mid \mathsf{bool} \mid \widehat{\tau} \times \widehat{\tau}' \mid \{\widehat{\tau}\}$$

We write $\widehat{\Gamma}$ for a typing context mapping variables to a-types. We lift the auxiliary a-value operations of erasure ($|\widehat{\tau}|$), annotation extraction ($\|\widehat{\tau}\|$), and compatibility ($\widehat{\tau}_1 \cong \widehat{\tau}_2$) to a-types in the obvious way.

We also define a "union" operation $\sqcup$ on compatible types as follows:

$$\omega_1^{\Phi_1} \sqcup \omega_2^{\Phi_2} = (\omega_1 \sqcup \omega_2)^{\Phi_1 \cup \Phi_2} \quad \mathsf{int} \sqcup \mathsf{int} = \mathsf{int} \quad \mathsf{bool} \sqcup \mathsf{bool} = \mathsf{bool}$$
$$(\widehat{\tau}_1 \times \widehat{\tau}_2) \sqcup (\widehat{\tau}_1' \times \widehat{\tau}_2') = (\widehat{\tau}_1 \sqcup \widehat{\tau}_1') \times (\widehat{\tau}_2 \sqcup \widehat{\tau}_2') \quad \{\widehat{\tau}\} \sqcup \{\widehat{\tau}'\} = \{\widehat{\tau} \sqcup \widehat{\tau}'\}$$

Finally, we write $\widehat{\tau} \sqsubseteq \widehat{\tau}'$ if $\widehat{\tau}' = \widehat{\tau} \sqcup \widehat{\tau}'$; this is essentially a subtyping relation.

$$\frac{}{\widehat{\Gamma} \vdash i : \mathsf{int} \;\&\; \emptyset} \qquad \frac{\widehat{\Gamma} \vdash e_1 : \mathsf{int} \;\&\; \Phi_1 \quad \widehat{\Gamma} \vdash e_2 : \mathsf{int} \;\&\; \Phi_2}{\widehat{\Gamma} \vdash e_1 + e_2 : \mathsf{int} \;\&\; \Phi_1 \cup \Phi_2} \qquad \frac{\widehat{\Gamma} \vdash e : \{\mathsf{int}^{\Phi_0}\} \;\&\; \Phi}{\widehat{\Gamma} \vdash \mathsf{sum}(e) : \mathsf{int} \;\&\; \Phi_0 \cup \Phi}$$

$$\frac{}{\widehat{\Gamma} \vdash b : \mathsf{bool} \;\&\; \emptyset} \quad \frac{\widehat{\Gamma} \vdash e : \mathsf{bool} \;\&\; \Phi}{\widehat{\Gamma} \vdash \neg e : \mathsf{bool} \;\&\; \Phi} \quad \frac{\widehat{\Gamma} \vdash e_1 : \mathsf{bool} \;\&\; \Phi_2 \quad \widehat{\Gamma} \vdash e_2 : \mathsf{bool} \;\&\; \Phi_2}{\widehat{\Gamma} \vdash e_1 \wedge e_2 : \mathsf{bool} \;\&\; \Phi_1 \cup \Phi_2}$$

$$\frac{\widehat{\Gamma} \vdash e_1 : \widehat{\tau}_1 \quad \widehat{\Gamma} \vdash e_2 : \widehat{\tau}_2}{\widehat{\Gamma} \vdash (e_1, e_2) : (\widehat{\tau}_1 \times \widehat{\tau}_2) \;\&\; \emptyset} \quad \frac{\widehat{\Gamma} \vdash e : \omega_1^{\Phi_1} \times \omega_2^{\Phi_2} \;\&\; \Phi}{\widehat{\Gamma} \vdash \pi_i(e) : \omega_i \;\&\; \Phi_i \cup \Phi}$$

$$\frac{\widehat{\Gamma} \vdash e_1 : \widehat{\tau}_1 \quad \widehat{\Gamma} \vdash e_2 : \widehat{\tau}_2 \quad \widehat{\tau}_1 \cong \widehat{\tau}_2}{\widehat{\Gamma} \vdash e_1 \approx e_2 : \mathsf{bool} \;\&\; \|\widehat{\tau}_1\| \cup \|\widehat{\tau}_2\|} \qquad \frac{\widehat{\Gamma} \vdash e_0 : \mathsf{bool} \;\&\; \Phi_0 \quad \widehat{\Gamma} \vdash e_1 : \widehat{\tau}_1 \quad \widehat{\Gamma} \vdash e_2 : \widehat{\tau}_2 \quad \omega_1 \cong \omega_2}{\widehat{\Gamma} \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : (\widehat{\tau}_1 \sqcup \widehat{\tau}_2)^{+\Phi_0}}$$

$$\frac{}{\widehat{\Gamma} \vdash \emptyset : \{\widehat{\tau}\} \;\&\; \emptyset} \quad \frac{\widehat{\Gamma} \vdash e : \widehat{\tau}}{\widehat{\Gamma} \vdash \{e\} : \{\widehat{\tau}\} \;\&\; \emptyset} \qquad \frac{\widehat{\Gamma} \vdash e_1 : \{\widehat{\tau}_1\} \;\&\; \Phi_1 \quad \widehat{\Gamma} \vdash e_2 : \{\widehat{\tau}_2\} \;\&\; \Phi_2 \quad \widehat{\tau}_1 \cong \widehat{\tau}_2}{\widehat{\Gamma} \vdash e_1 \cup e_2 : \{\widehat{\tau}_1 \sqcup \widehat{\tau}_2\} \;\&\; \Phi_1 \cup \Phi_2}$$

$$\frac{\widehat{\Gamma} \vdash e_1 : \{\widehat{\tau}_1\} \;\&\; \Phi_1 \quad \widehat{\Gamma} \vdash e_2 : \{\widehat{\tau}_2\} \;\&\; \Phi_2 \quad \widehat{\tau}_1 \cong \widehat{\tau}_2}{\widehat{\Gamma} \vdash e_1 - e_2 : \{\widehat{\tau}_1\} \;\&\; \|\{\widehat{\tau}_1\}^{\Phi_1}\| \cup \|\{\widehat{\tau}_2\}^{\Phi_2}\|} \qquad \frac{\widehat{\Gamma} \vdash e : \{\{\widehat{\tau}\}^{\Phi_2}\} \;\&\; \Phi_1}{\widehat{\Gamma} \vdash \bigcup e : \{\widehat{\tau}\} \;\&\; \Phi_1 \cup \Phi_2}$$

$$\frac{\widehat{\Gamma} \vdash e_1 : \{\widehat{\tau}_1\} \;\&\; \Phi_1 \quad \widehat{\Gamma}, x{:}\widehat{\tau}_1 \vdash e_2 : \omega \;\&\; \Phi_2}{\widehat{\Gamma} \vdash \{e_2 \mid x \in e_1\} : \{\omega_2^{\Phi}\} \;\&\; \Phi_1} \qquad \frac{x{:}\widehat{\tau} \in \widehat{\Gamma}}{\widehat{\Gamma} \vdash x : \widehat{\tau}} \qquad \frac{\widehat{\Gamma} \vdash e_1 : \widehat{\tau}_1 \quad \widehat{\Gamma}, x : \widehat{\tau}_1 \vdash e_2 : \widehat{\tau}_2}{\widehat{\Gamma} \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \widehat{\tau}_2}$$

Fig. 8. Type-based provenance analysis

We interpret a-types $\widehat{\tau}$ as sets $\mathcal{A}[\![\widehat{\tau}]\!]$ of a-values. We interpret the annotations in a-types as upper bounds on the annotations in the corresponding a-values:

$$\begin{aligned} \mathcal{A}[\![\mathsf{int}]\!] &= \{i \mid i \in \mathbb{Z}\} & \mathcal{A}[\![\widehat{\tau}_1 \times \widehat{\tau}_2]\!] &= \{(v_1, v_2) \mid v_1 \in \mathcal{A}[\![\widehat{\tau}_1]\!], v_2 \in \mathcal{A}[\![\widehat{\tau}_2]\!]\} \\ \mathcal{A}[\![\mathsf{bool}]\!] &= \{b \mid b \in \mathbb{B}\} & \mathcal{A}[\![\{\widehat{\tau}\}]\!] &= \{V \mid \forall v \in V. v \in \mathcal{A}[\![\widehat{\tau}]\!]\} \\ \mathcal{A}[\![\omega^{\Phi}]\!] &= \{w^{\Psi} \mid \Psi \subseteq \Phi, w \in \mathcal{A}[\![\omega]\!]\} \end{aligned}$$

The syntactic operations $|-|$, $\|-\|$, $\sqsubseteq$ and $\sqcup$ correspond to appropriate semantic operations on sets of a-values. We note some useful properties of these operations:

Lemma 2. *1. If $v \in \mathcal{A}[\![\widehat{\tau}]\!]$ then $|v| \in \mathcal{T}[\![|\widehat{\tau}|]\!]$ and $\|v\| \subseteq \|\widehat{\tau}\|$.*
2. If $\widehat{\tau}_1 \sqsubseteq \widehat{\tau}_2$ then $\mathcal{A}[\![\widehat{\tau}_1]\!] \subseteq \mathcal{A}[\![\widehat{\tau}_2]\!]$ and $\|\widehat{\tau}_1\| \subseteq \|\widehat{\tau}_2\|$.
3. If $\widehat{\tau}_1 \sqcup \widehat{\tau}_2$ is defined then $\mathcal{A}[\![\widehat{\tau}_1 \sqcup \widehat{\tau}_2]\!] = \mathcal{A}[\![\widehat{\tau}_1]\!] \cup \mathcal{A}[\![\widehat{\tau}_2]\!]$ and $\|\widehat{\tau}_1 \sqcup \widehat{\tau}_2\| = \|\widehat{\tau}_1\| \cup \|\widehat{\tau}_2\|$.

The annotated typing judgment $\widehat{\Gamma} \vdash e : \widehat{\tau}$ (sometimes written $\widehat{\Gamma} \vdash e : \omega \;\&\; \Phi$ for readability, provided $\widehat{\tau} = \omega^{\Phi}$) extends the plain typing judgment shown in Figure 2.

Proposition 2. *The judgment $\Gamma \vdash e : \tau$ is derivable if and only if for any $\widehat{\Gamma}$ enriching Γ, there exists a $\widehat{\tau}$ enriching τ such that $\widehat{\Gamma} \vdash e : \widehat{\tau}$. Moreover, given $\Gamma \vdash e : \tau$, and $\widehat{\Gamma}$ enriching Γ, we can compute $\widehat{\tau}$ in polynomial time (by a simple syntax-directed algorithm).*

The correctness of the analysis is proved with respect to the provenance-tracking semantics. This property takes the form of a type-soundness theorem: we simply need to show that if the input environment $\widehat{\gamma}$ is well-formed at annotated context $\widehat{\Gamma}$ then the result $\mathcal{P}[\![e]\!]\widehat{\gamma}$ is well-formed at type $\widehat{\tau}$:

(a)

$$\widehat{\Gamma} = [R : \{(A : \mathsf{int}^a, B : \mathsf{int}^b)\}, S : \{(C : \mathsf{int}^c, D : \mathsf{int}^d, E : \mathsf{int}^e)\}]$$

(b)

$$\begin{array}{ll}
\widehat{\Gamma} \vdash \Pi_A(R) & : \{(A : \mathsf{int}^a)\} \\
\widehat{\Gamma} \vdash \sigma_{A=B}(R) & : \{(A : \mathsf{int}^a, B : \mathsf{int}^b)\}^{a,b} \\
\widehat{\Gamma} \vdash R \times S & : \{(A : \mathsf{int}^a, B : \mathsf{int}^b, C : \mathsf{int}^c, D : \mathsf{int}^d, E : \mathsf{int}^e)\} \\
\widehat{\Gamma} \vdash \Pi_{BE}(\sigma_{A=D}(R \times S)) & : \{(B : \mathsf{int}^b, E : \mathsf{int}^e)\}^{a,d} \\
\widehat{\Gamma} \vdash R \cup \rho_{A/C,B/D}(\Pi_{CD}(S)) & : \{(A : \mathsf{int}^{a,c}, B : \mathsf{int}^{b,d})\} \\
\widehat{\Gamma} \vdash R - \rho_{A/D,B/E}(\Pi_{DE}(S)) & : \{(A : \mathsf{int}^a, B : \mathsf{int}^b)\}^{a,b,d,e} \\
\widehat{\Gamma} \vdash \mathsf{sum}(\Pi_A(R)) & : \mathsf{int}^a \\
\widehat{\Gamma} \vdash \mathsf{count}(R) & : \mathsf{int}
\end{array}$$

Fig. 9. (a) Annotated input context (b) Examples of provenance analysis

Theorem 2. *If* $\widehat{\Gamma} \vdash e : \widehat{\tau}$ *then* $\mathcal{P}[\![e]\!] : \mathcal{A}[\![\widehat{\Gamma}]\!] \to \mathcal{A}[\![\widehat{\tau}]\!]$.

This theorem tells us that the annotations we obtain (statically) by provenance analysis over-approximate those obtained (dynamically) by provenance tracking provided the initial value $\widehat{\gamma}$ matches $\mathcal{A}[\![\widehat{\Gamma}]\!]$.

Example 4. Consider an annotated type context $\widehat{\Gamma}$, shown in Figure 9(a), where we have annotated field values A, B, C, D, E with colors a, b, c, d, e respectively. Figure 9(b) shows the results of static analysis for the queries in Figure 7. In some cases, the type information simply reflects the field names which are present in the output. However, the colors are not affected by renamings, as in $\rho_{A/C,B/D}$. Furthermore, note that (if we replace the colors a, b, c, d, e with color sets $\{a_1, a_2, a_3\}$, etc.) in each case the type-level colors over-approximate the value-level colors calculated in Figure 7.

Example 5. To illustrate how the analysis works in practice, we consider an extended example for a query that performs grouping and aggregation:

$$Q(R) = \{(\pi_1(x), \mathsf{sum}(G(x))) \mid x \in R\}$$

where we make the following abbreviations:

$$\begin{array}{ll}
G(x) := \bigcup\{\mathsf{if}\ \pi_1(y) \approx \pi_1(x)\ \mathsf{then}\ \{\pi_2(y)\}\ \mathsf{else}\ \{\} \mid y \in R\} & \\
\widehat{\tau}_R := \mathsf{int}^a \times \mathsf{int}^b & \widehat{\Gamma} := R{:}\{\widehat{\tau}_R\} \\
\widehat{\Gamma}_1 := \widehat{\Gamma}, x{:}\widehat{\tau}_R & \widehat{\Gamma}_2 := \widehat{\Gamma}_1, y{:}\widehat{\tau}_R
\end{array}$$

We will derive $\widehat{\Gamma} \vdash Q(R) : \{\mathsf{int}^a \times \mathsf{int}^{a,b}\}$. The derivation illustrates how color a is propagated from the to both parts of the result type, while color b is only propagated to the second column.

First, we can reduce the analysis of Q to analyzing G as follows:

$$\dfrac{\widehat{\Gamma} \vdash R : \{\widehat{\tau}_R\} \qquad \dfrac{\dfrac{\widehat{\Gamma}_1 \vdash x : \widehat{\tau}_R}{\widehat{\Gamma}_1 \vdash \pi_1(x) : \mathsf{int}^a} \quad \dfrac{\widehat{\Gamma}_1 \vdash G(x) : \{\mathsf{int}^b\}^a}{\widehat{\Gamma}_1 \vdash \mathsf{sum}(G(x)) : \mathsf{int}^{a,b}}}{\widehat{\Gamma}_1 \vdash (\pi_1(x), \mathsf{sum}(G(x))) : \mathsf{int}^a \times \mathsf{int}^{a,b}}}{\widehat{\Gamma} \vdash \{(\pi_1(x), \mathsf{sum}(G(x))) \mid x \in R\} : \{\mathsf{int}^a \times \mathsf{int}^{a,b}\}}$$

We next reduce the analysis of G to an analysis of the conditional inside G:

$$\dfrac{\dfrac{\dfrac{\widehat{\Gamma}_1 \vdash R : \{\widehat{\tau}_R\} \quad \widehat{\Gamma}_2 \vdash \mathsf{if}\ \pi_1(y) \approx \pi_1(x)\ \mathsf{then}\ \{\pi_2(y)\}\ \mathsf{else}\ \{\} : \{\mathsf{int}^b\}^a}{\widehat{\Gamma}_1 \vdash \{\mathsf{if}\ \pi_1(y) \approx \pi_1(x)\ \mathsf{then}\ \{\pi_2(y)\}\ \mathsf{else}\ \{\} \mid y \in R\} : \{\{\mathsf{int}^b\}^a\}}}{\widehat{\Gamma}_1 \vdash \bigcup\{\mathsf{if}\ \pi_1(y) \approx \pi_1(x)\ \mathsf{then}\ \{\pi_2(y)\}\ \mathsf{else}\ \{\} \mid y \in R\} : \{\mathsf{int}^b\}^a}$$

Finally, we can analyze the conditional as follows:

$$\dfrac{\dfrac{\dfrac{\overline{\widehat{\Gamma}_2 \vdash y : \widehat{\tau}_R}}{\widehat{\Gamma}_2 \vdash \pi_1(y) : \mathsf{int}^a} \quad \dfrac{\overline{\widehat{\Gamma}_2 \vdash y : \widehat{\tau}_R}}{\widehat{\Gamma}_2 \vdash \pi_1(x) : \mathsf{int}^a}}{\widehat{\Gamma}_2 \vdash \pi_1(y) \approx \pi_1(x) : \mathsf{bool}^a} \quad \dfrac{\dfrac{\overline{\widehat{\Gamma}_2 \vdash y : \widehat{\tau}_R}}{\widehat{\Gamma}_2 \vdash \pi_2(y) : \mathsf{int}^b}}{\widehat{\Gamma}_2 \vdash \{\pi_2(y)\} : \{\mathsf{int}^b\}} \quad \dfrac{}{\widehat{\Gamma}_2 \vdash \{\} : \{\mathsf{int}\}}}{\widehat{\Gamma}_2 \vdash \mathsf{if}\ \pi_1(y) \approx \pi_1(x)\ \mathsf{then}\ \{\pi_2(y)\}\ \mathsf{else}\ \{\} : \{\mathsf{int}^b\}^a}$$

3.3 Discussion

We have implemented a prototype interpreter for the NRC that performs ordinary typechecking and evaluation as well as provenance tracking and analysis. We used this implementation to construct the examples.

We chose to study provenance via the NRC because it is a clean and system-independent model; we believe our results can be specialized to common database implementations and physical operators without much difficulty. We have not yet investigated scaling this approach to large datasets. There are several apparent obstacles to implementing annotation-based provenance tracking in standard database systems that do not natively support annotation. Recent research has begun to address this problem [6, 18] and we plan to investigate whether these techniques can be used to implement our approach.

Static provenance analysis is also more expensive than ordinary typechecking, but since the overhead is proportional only to the size of the *query*, not the (usually much larger) data, this seems acceptable. Moreover, static analysis may be useful in optimizing provenance tracking, for example by using the results of static analysis to avoid tracking annotations that are statically irrelevant to the output.

4 Related and Future Work

Slicing and other dependence analyses Dependence tracking and analysis have been shown to be useful in many contexts [1] such as program slicing [7, 16, 24], memoization and caching [2, 4], and information-flow security [20]. In program slicing [24],

the goal is to identify a (small) set of program points whose execution contributes to the value of an output variable (or other observable behavior). This is analogous to our approach to provenance, except that provenance identifies relevant parts of the *input database*, not the *program* (i.e. query). Our approach is inspired by, and in some cases could be viewed as an adaptation of, these techniques to a database setting with collection types.

Provenance in databases Most work on provenance in databases [23, 15, 11, 12, 10] has focused on identifying information that explains *why* some data is present in the output of a query (or view) or *where* some data in the output was copied from in the input. However, semantic characterizations of these intuitions have been elusive and difficult to generalize beyond monotone relational queries. Our work generalizes some of these techniques and provides clear semantic guarantees and qualitatively useful provenance information in the presence of grouping and aggregation.

Updates Some recent work has generalized where-provenance to database updates [9, 10], motivated by "curated" scientific databases that are updated frequently, often by (manual) copying from other sources. Our approach addresses an orthogonal issue; we plan to investigate dependency provenance for updates.

Workflow provenance Provenance has also been studied in geospatial and scientific "grid" computation [8, 17, 21], particularly for *workflows* (visual programs written by scientists). At present, formal correctness criteria have not been identified for most of these approaches, but we believe it to be worthwhile to seek a foundation for workflow provenance using dependence analysis.

Annotations Recent research on annotations, uncertainty, and incomplete information [6, 5, 18, 19] is closely related to provenance. Green et al. [19] showed that relations with semiring-valued annotations generalize several variations of the relational model, including set, bag, probabilistic, and incomplete information models, and identified a relationship between free semiring-valued relations and why-provenance.

5 Conclusions

Provenance information that relates parts of the result of a query to "relevant" parts of the input is useful for many purposes, including judging the reliability of information based on the relevant sources and identifying parts of the database that may be responsible for an error in the output of a query. We have argued that the notion of *dependence*, familiar from program slicing, information flow security, and other analyses, provides a solid semantic foundation for understanding provenance for complex database queries. In this paper we introduced a semantic characterization of *dependency provenance*, showed that minimal dependency provenance is not computable, and presented approximate tracking and analysis techniques. We believe there are many promising directions for future work, including implementing efficient practical techniques, identifying more sophisticated and useful dependency properties, and studying dependency provenance in other settings such as update languages and workflows.

Acknowledgments We wish to thank Peter Buneman and Stijn Vansummeren for helpful discussions on this work.

References

1. Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *POPL*, pages 147–160. ACM Press, 1999.
2. Martín Abadi, Butler Lampson, and Jean-Jacques Lévy. Analysis and caching of dependencies. In *ICFP*, pages 83–91, New York, NY, USA, 1996. ACM Press.
3. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
4. Umut A. Acar, Guy E. Blelloch, and Robert Harper. Selective memoization. In *Proceedings of the 30th Annual ACM Symposium on Principles of Programming Languages*, 2003.
5. Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.
6. Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. *VLDB Journal*, 14(4):373–396, 2005.
7. S. Biswas. *Dynamic Slicing in Higher-Order Programming Languages*. PhD thesis, University of Pennsylvania, 1997.
8. Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, 2005.
9. Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *SIGMOD 2006*, pages 539–550, 2006.
10. 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.
11. Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and where: A characterization of data provenance. In *ICDT 2001*, 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. Peter Buneman, Shamim A. Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with complex objects and collection types. *Theor. Comp. Sci.*, 149(1):3–48, 1995.
14. James Cheney, Amal Ahmed, and Umut Acar. Provenance as dependency analysis. Technical Report arXiv:0708.2173v1, arXiv.org e-Print archive, 2007.
15. 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.
16. John Field and Frank Tip. Dynamic dependence in term rewriting systems and its application to program slicing. *Information and Software Technology*, 40(11–12):609–636, November/December 1998.
17. Ian Foster and Luc Moreau, editors. *Proceedings of the 2006 International Provenance and Annotation Workshop (IPAW 2006)*. Number 4145 in LNCS. Springer-Verlag, 2006.
18. Floris Geerts, Anastasios Kementsietsidis, and Diego Milano. Mondrian: Annotating and querying databases through colors and blocks. In *ICDE 2006*, page 82, 2006.
19. Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, New York, NY, USA, 2007. ACM Press.
20. Andrei Sabelfeld and Andrew Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
21. Yogesh Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.
22. P. Wadler. Comprehending monads. *Mathematical Structures in Computer Science*, 2:461–493, 1992.
23. Y. Richard Wang and Stuart E. Madnick. A polygen model for heterogeneous database systems: The source tagging perspective. In *VLDB*, pages 519–538, 1990.
24. Mark Weiser. Program slicing. In *ICSE*, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.