A Model for Fine-Grained Data Citation

Susan B. Davidson
University of Pennsylvania
Philadelphia, PA
susan@cis.upenn.edu

Daniel Deutch
Tel Aviv University
Tel Aviv, Israel
danielde@post.tau.ac.il

Tova Milo
Tel Aviv University
Tel Aviv, Israel
milo@post.tau.ac.il

Gianmaria Silvello
University of Padua
Padua, Italy
silvello@dei.unipd.it

ABSTRACT

An increasing amount of information is being collected in structured, evolving, curated databases, driving the question of how information extracted from such datasets via queries should be cited. Unlike traditional research products, such books and journals, which have a fixed granularity, data citation is a challenge because the granularity varies. Different portions of the database, with varying granularity, may have different citations. Furthermore, there are an infinite number of queries over a database, each accessing and generating different subsets of the database, so we cannot hope to explicitly attach a citation to every possible result set and/or query. We present the novel problem of *automatically generating citations for general queries* over a relational database, and explore a solution based on a set of *citation views*, each of which attaches a citation to a view of the database. Citation views are then used to automatically construct citations for general queries. Our approach draws inspiration from results in two areas, *query rewriting using views* and *database provenance* and combines them in a robust model. We then discuss open issues in developing a practical solution to this challenging problem.

1. INTRODUCTION

Citation is essential to traditional scholarship. It helps identify the cited material so that it can be retrieved, gives credit to the creator of the material, dates it, and so on. In the context of printed materials, such as books and journals, citation is well understood. However, the world is now digital, and an increasing amount of information is being collected in structured and evolving curated databases, driving database owners, publishers and standards groups to consider how such data should be cited.

Unlike traditional publications which have a fixed granularity to which citations can be attached – e.g. a conference proceedings, or a paper in a conference proceedings – data citation is a challenge because the granularity varies. For example, the content of a scientific database frequently represents the effort of a large number of members of the scientific community. Different portions of the database, with varying granularity, are contributed and/or curated by different subgroups of these individuals. While a citation to the database as a whole is typically provided as a traditional publication (e.g. in the annual NAR Database Issue), whose author list includes the owners and developers of the resource, there is a growing belief that contributors/curators should be acknowledged when their data is extracted and used. That is, when the database is queried and a result set returned, the citation for that data should include information about the contributors/curators of the result set as well as of the data used to compute it. The latter depends on the query.

Since there are a potentially infinite number of queries, each accessing and generating different subsets of the database, we cannot hope to explicitly attach a citation to every possible result set and/or query. Instead, we must find ways of specifying citations for some semantically meaningful portions of the database (possibly defined declaratively via queries), and use these to automatically construct citations for more general queries. Thus data citation is a *computational* problem, as argued in [3] and fleshed out in more detail here.

An interesting example of data citation, which we will refer to throughout the remainder of this paper, is the IUPHAR/BPS Guide to Pharmacology (GtoPdb) [8].[1] GtoPdb is a relational database that contains expertly curated information about drugs in clinical use and some experimental drugs, together with information on the cellular targets of the drugs and their mechanisms of action in the body.

Users view information through a hierarchy of web pages: The top level divides information by families of drug targets that reflect typical pharmacological thinking; lower levels divide the families into sub-families and so on down to individual drug targets and drugs. The content of a particular family "landing" page is curated by a committee of experts; a family may also have a "detailed introduction page" which is written by a set of contributors, who are not necessarily the same as the committee of experts for the family.

The citations for these views of the database, which are parameterized by the family id, therefore vary: the citation for GtoPdb as a whole is a traditional paper written by the database owners, a citation to a family page includes the committee members who curated the content, and a citation

[1] http://www.guidetopharmacology.org/

8th Biennial Conference on Innovative Data Systems Research (CIDR '17). January 8-11, 2017, Chaminade, California, USA.

to a family detailed introduction page includes the contributors who wrote the content. Currently, citations for these views are hard-coded into the web pages – queries are issued against the underlying relational database to obtain content for the page as well as the committee members or contributors. Thus, GtoPdb in fact does generates citations, but only to a subset of the possible queries against the underlying relational database, i.e. those corresponding to web-page views of the data.

In the future, the owners of GtoPdb would like to allow users to issue *general queries* against the relational database and automatically generate a citation for the result. This novel *computational* problem is what we address in the remainder of the paper.

Our approach draws inspiration from results in two areas: *query rewriting using views*, which has been well studied in the context of query optimization, maintenance of physical data independence, and data integration [2, 6, 7]; and *database provenance* [4, 5].

We start with a notion of *citation views*, through which database owners specify how citations are attached to the output of some number of queries (views) over the database representing typical usage patterns. The citation to a general query is then constructed by *rewriting* it to a set of equivalent queries using the citation views, and combining the citations attached to these views to construct a citation to the query (Section 2).

To construct the citation, we leverage the fact that citations and provenance are both forms of *annotation* that are manipulated through queries [4]. In particular, the *joint* $(\cdot)$ and *alternative* $(+)$ use of annotations within a rewriting are modeled using the *semirings* approach of [5]; the model is then extended to capture combinations of *all alternative rewritings* $(+^R)$. As a consequence, the citations associated with output tuples of a given query are independent of the actual query rewriting employed by the optimizer. Finally, the size of the resulting citation is reduced by lifting citation annotations from individual tuples to sets of tuples via an *aggregation* function (Agg). The database owner specifies a *policy* by which citations to general queries are constructed by choosing an interpretation of the combining functions $+$, $\cdot$, $+^R$, and Agg. The *citation semiring* model, together with several natural interpretations of the combining functions, is presented in Section 3. We close in Section 4 by describing the implications for database owners, and discussing several open issues that must be addressed in developing a practical solution to this challenging problem.

2. CITATION VIEWS AND REWRITING

2.1 Citation Views

We consider a relational setting with queries and views expressed as Conjunctive Queries (see [1] for an overview).

Given a database D, we start with a set of *view definitions*. Each view definition has an associated *citation query* C_V and *citation function* F_V. A *citation view* consists of a view definition, and its citation query and function. The view and query are *parameterized*, by the same set of parameters. Intuitively for each valuation to the parameters, the citation function defines a single citation for all tuples in the instantiation of the view with the given valuation.

More formally:

DEFINITION 2.1. *A* citation view *is a triple (V, C_V, F_V) where*

- *V is the* view definition *of form $\lambda X.V(Y) : -Q$;*
- *C_V is the* citation query *of form $\lambda X.\ C_V(Y') : -Q'$; and*
- *F_V is the* citation function *which transforms the output of the citation query into a citation in some desired format, such as JSON or XML.*

In V and C_V:

- *Q and Q' are conjunctions of atoms;*
- *$X = [x_1, ..., x_n]$, $Y = [y_1, ..., y_m]$, $Y' = [y'_1, ..., y'_p]$ are ordered sequences of variables;*
- *$X \subseteq Y$ are subsets of the variables in Q;*
- *X and Y' are subsets of the variables in Q'.*

The terms $x_1, ..., x_n$ are referred to as the parameters *of V and C_V. Parameters are optional, in which case the λ-term may be dropped. If there is a λ-term the citation view is called* parameterized.

Intuitively, a citation view is defined by a query Q which is parameterized by X. For a given database instance, each choice of values for the parameters may lead to a different view output; we denote the view instance by applying the view to the input parameter values, e.g. if the view is

$$\lambda(x_1, \ldots, x_n)V(Y) : -Q$$

and is passed parameter values $(a_1, \ldots, a_n)$, we write

$$V(Y)(a_1, \ldots, a_n).$$

Since the citation query Q' is also parametrized by X, each valuation of X leads to a different result for Q'. The output of the citation function – the *citation* for the view – becomes an annotation on each tuple in the output of the instantiated view. Thus for every sequence of parameter values $(a_1, \ldots, a_n)$, the citation for all tuples in $V(Y)(a_1, \ldots, a_n)$ is $F_v(C_v(Y')(a_1, \ldots, a_n))$.

EXAMPLE 2.1. *We use as a running example a simplified database schema for GtoPdb, in which keys are underlined:*

Family(FID, FName, Type)
FamilyIntro(FID, Text)
Person(PID, PName, Affiliation)
FC(FID, PID), FID references Family,
PID references Person
FIC (FID, PID), FID references FamilyIntro,
PID references Person
MetaData(Type, Value)

Intuitively, FC captures the committee members who curate the content of a family page while FIC captures the contributors who author the Family Introduction page of a family. The last table, MetaData, captures other information that may be useful to include in citations, such as the owner of the database ("Owner", "Tony Harmar"), the URL of the database ("URL", "guidetopharmacology.org") and the current version number of the database ("Version", "23").

To express citation views for GtoPdb we start by defining a set of view definitions.

$\lambda F.V1(F,N,Ty) \quad :-Family(F,N,Ty)$
$\lambda F.V2(F,Tx) \quad :-FamilyIntro(F,Tx)$
$V3(F,N,Ty) \quad :-Family(F,N,Ty)$
$\lambda Ty.V4(F,N,Ty) \quad :-Family(F,N,Ty)$
$\lambda Ty.V5(F,N,Ty,Tx) :-Family(F,N,Ty), FamilyIntro(F,Tx)$

All views except $V3$ are parameterized. $V1$ and $V2$ restrict the output to a single tuple since the parameter, F, corresponds to the key FID in Family; $V4$ and $V5$ restrict the output to a subset of tuples. $V3$ contains all tuples in Family.

For each of the views, we define a citation query.

$\lambda F.\ C_{V1}(F,N,Pn) :- Family(F,N,Ty), FC(F,C), Person(C,Pn,A)$

$\lambda F.\ C_{V2}(F,N,Tx,Pn) :- Family(F,N,Ty), FamilyIntro(F,Tx), FIC(F,C), Person(C,Pn,A)$

$C_{V3}(X1,X2) :- MetaData(T1,X1), T1 = \text{“}Owner\text{”}, MetaData(T2,X2), T2 = \text{“}URL\text{”}$

$\lambda Ty.\ C_{V4}(Ty,N,Pn) :- Family(F,N,Ty), FC(F,C), Person(C,Pn,A)$

$\lambda Ty.\ C_{V5}(N,Ty,Tx,Pn) :- Family(F,N,Ty), FamilyIntro(F,Tx), FIC(F,C), Person(C,Pn,A)$

Since V1 and its citation query C_{V1} are parameterized by F, each valuation of V1 results in a single tuple and each tuple may have a different citation. As an example, for the family with FID=11 the citation is a function of the result of C_{V1}(F,N,Pn)(“11”), which consists of the family id, name and list of committee members. The citation function F_{V1} could format this result in JSON as:
{ID: “11”, Name: “Calcitonin”, Committee: [“Hay”, “Poyner”]}

Similarly, V2 and its citation function C_{V2} are parameterized by F, resulting in a potentially different citation for each tuple. For the family with FID=11 the citation is a function of the result of C_{V2}(F,N,Tx, Pn)(“11”). F_{V2} could format this result in JSON as: {ID: “11”, Name: “Calcitonin”, Text: “The calcitonin peptide family”, Contributors: [“Brown”, “Smith”]}

In contrast to $V1$ and $V2$, $V3$ is not parameterized, which means that all tuples have the same citation in V3. C_{V3} uses the MetaData table to obtain the url and owner of the database. F_{V3} could format this result in JSON as:
{URL: “guidetopharmacology.org”, Owner: “Tony Harmar”}

The parameter of $V4$ does not correspond to a key. This means that a single citation will be produced for all tuples with the same type. For instance, the citation for the view tuples $V4(F,N,Ty)$(“gpcr”)) could be
{Type: “gpcr”, Contributors: [{Name:“Calcitonin”, Committee:[“Hay”, “Poyner”]}, {Name:“Calcium-sensing”, Committee:[“Bilke”, “Conigrave”, “Shoback”]}]}

$V5$ produces a citation for all the introductions of the families with the same type. The main difference between $V4$ and $V5$ is that $V4$ credits the committee members of families, whereas $V5$ credits the contributors who wrote the introductions.

For instance, the citation for the view tuples $V5(N,Ty,Tx,Pn)$(“gpcr”)) could be:
{Type: “gpcr”, Contributors: [{Name:“Calcitonin”, Committee:[“Nichols”, “Palmer”]}, {Name:“Orexin”, Committee:[“Alda”, “Palmer”]}]}

2.2 Query Rewritings

Database owners specify a set of citation views, from which the citation for a general query over the database will be constructed. Our approach is to rewrite as much of the query as possible using the view definitions, and combine their citations to construct a citation for the input query. We start by defining what a rewriting is.

DEFINITION 2.2. *Let $\mathcal{R}$ be a set of relation names, Q be a query, and $\mathcal{V}$ be a set of views. The query Q' is a* rewriting *of Q using $\mathcal{V}$ if:*

- *the subgoals of Q' are either relation names in $\mathcal{R}$, views in $\mathcal{V}$, or comparison predicates;*
- *Q' is equivalent to Q;*
- *no subgoal of Q' can be removed and obtain an equivalent query; and*
- *no subset of subgoals of Q' can be replaced by a view in $\mathcal{V}$ and obtain an equivalent query.*

A rewriting is *total* if its subgoals contain only views and comparison predicates; otherwise, if its subgoals also contain relation names, it is *partial*.

We illustrate the trade-offs between different rewritings in terms of the citations generated through two examples.

EXAMPLE 2.2. *Consider the following query which finds the names of all gpcr families that have a detailed introduction page:*

$Q(N) :- Family(F,N,Ty), Ty = \text{“}gpcr\text{”}, FamilyIntro(F,Tx)$

Q can be rewritten using either $V1$ and $V2$ or $V4$ and $V2$:

$Q_1(N) :- V1(F,N,Ty), Ty = \text{“}gpcr\text{”}, V2(F,Tx)$
$Q_2(N) :- V4(F,N,Ty)(\text{“}gpcr\text{”}), V2(F,Tx)$

Both Q_1 and Q_2 are total rewritings, however in Q_1 there is a remaining comparison predicate. The main difference between $V1$ and $V4$ is that $Q1$ uses $V1$ while $Q2$ uses $V4$; the citation query C_{V4} is parameterized by the family type Ty, while C_{V1} is not.

Q_2 leads to a more specific citation than Q_1 because the comparison predicate Ty = “gpcr” in Q matches the lambda term of $V4$ and can therefore be absorbed as passing in a parameter value, V4(F,N,Ty)(“gpcr”). This groups together all tuples sharing the type “gpcr”, yielding a single citation as described in Example 2.1. In contrast, $Q1$ would yield a citation for each distinct gpcr family, since its lamba term is F, which corresponds to the family id.

EXAMPLE 2.3. *As another example, consider a query which finds the name and text of the introduction of families with type “gpcr”.*

$Q(N,Tx) :- Family(F,N,Ty), FamilyIntro(F,Tx), Ty = \text{“}gpcr\text{”}$

Q can be rewritten in several ways, including:

$Q_1(N, Tx) : -V1(F, N, Ty), V2(F, Tx), Ty =$ "*gpcr*"
$Q_2(N, Tx) : -V3(F, N, Ty), V2(F, Tx), Ty =$ "*gpcr*"
$Q_3(N, Tx) : -V4(F, N, Ty)($"*gpcr*"$), V2(F, Tx)$
$Q_4(N, Tx) : -V5(F, N, Ty, Tx)($"*gpcr*"$)$

All of these rewritings are total, however Q_1, Q_2 and Q_3 use two views, whereas Q_4 uses only one view. The difference between Q_1 and Q_2 is that the former uses V_1, while the latter uses V_3; C_{V1} produces a citation for each distinct family, whereas C_{V3} provides a single citation for the entire view. In this sense, $V3$ is more general than $V1$. In neither case is the comparison predicate $Ty =$ "gpcr" matched by the lambda term of the view.

Both Q_3 and Q_4 use views whose lambda term (λTy) matches the comparison predicate. They are therefore more specific than Q_1 and Q_2. The difference between Q_3 and Q_4 is that Q_4 uses one view whereas Q_3 uses two.

Overall, we might prefer Q_4 to the other rewritings because: (i) it is a total rewriting; (ii) it uses the smallest number of views; and (iii) the comparison predicate of the query is matched by the lambda term of the view, so there are no remaining non-view predicates.

2.3 Preference Model

As demonstrated above, a single query may be rewritten in multiple ways using a set of views. How should we define the citation for the query result as a function of these rewritings? A first observation is that some rewritings are preferable to others in the sense that they lead to citations that are more precise. For instance, if a query may be totally re-written using a single view, then the most accurate citation would be the citation for that view. More generally, plausible criteria could aim at minimizing one or more of the following:

- We prefer rewritings that cover as many terms of the query as possible using views, leaving a minimal number of terms "uncovered", i.e. captured by directly accessing base relations or appearing as comparison predicates. In other words, total rewritings are preferable to partial ones, and total rewritings with no remaining comparison predicates are preferable to those with remaining comparison predicates, as illustrated in Example 2.2. Such rewritings lead to citations that are more "comprehensive".
- We prefer rewritings that use the smallest number of views. For example, the total rewriting Q_4 in Example 2.3 is preferable to Q_3. Such rewritings lead to citations that are more "focused" and compact.

In Section 3.4, these ideas will be generalized to an *order relation* over rewritings.

We now define how to associate a citation with a query result based on the possible rewritings and the citation views used therein.

3. COMBINING CITATIONS

Having defined citations for views, we must now combine them to form a citation for the query result. To do this, observe that citations and provenance are both forms of *annotation* that are manipulated through queries [4]. We therefore take inspiration from work on database provenance, in particular that of *provenance semirings* [5], to model the different ways in which citations views are combined.

3.1 Semiring Model

In provenance semirings, provenance tokens (base annotations) are associated with each tuple in a relational instance (EDB). Restricting our attention to SPJU queries, there are two ways in which tuples are combined through queries:

- *Joint use*, as in a join which combines two tuples to form a new tuple. In this case, the provenance annotation of the new tuple is the '$\cdot$' of the annotations of the two input tuples.
- *Alternate use*, as when one or more tuples are "identified" via unions or projections to form a new tuple. In this case, the provenance annotation of the new tuple is the '$+$' of the annotations of all input tuples.

In citations, annotations are defined through the citation queries (and their corresponding citation functions). Citation views are combined in two different ways when providing a citation to general queries: they may appear *jointly* in an equivalent rewriting of the query, or they may appear in *alternate* rewritings of the query.

In the following, we will assign query output with citations that are the combination of results of citation functions, through $+$ and $\cdot$ based on the use of the views in the query. The resulting structure of citations is that of a *commutative semiring*: we start with a set of basic citations C, and introduce an abstract operation $+$ on it with the properties that $+$ is commutative, associative, and has some neutral element 0 in C. Similarly we introduce an operation $\cdot$ with the same properties, but with a different neutral element 1. Last, we impose that $\cdot$ is distributive over $+$. The resulting structure $(C, +, \cdot, 0, 1)$ is a commutative semiring.

3.2 Citation Semirings

We start by defining a citation for a *single binding* of a *single rewriting* of the query. This dictates a single output tuple, and a particular valuation to the parameters of the views. We define the citation of the output tuples as the joint use of citations for the views and the valuations to their parameters, denoted by ".".

DEFINITION 3.1. *Let Q be a query, let V be a set of citation views and let Q' be a (partial) rewriting of Q using $V1, ..., Vn \in V$. Further let B be a binding to the variables of Q', yielding an output tuple t. The citation for t w.r.t. Q, Q', V, B, denoted as $cite(t, Q, Q', V, B)$, is defined as*
$F_{V1}(C_{V1}(B1)) \cdot \ldots \cdot F_{Vn}(C_{Vn}(Bn))$
where Bi is the application of B to the variables occurring in an atom involving Vi in Q'.

EXAMPLE 3.1. *Consider the rewriting Q_1 from Example 2.2, and consider the binding to its variables F="11", N="Calcitonin", Ty = "gpcr" and Tx="The calcitonin peptide family". The resulting tuple is ("Calcitonin"), and the citation we get* for this particular binding *is the citation assigned in $V1$ to family "11" (note that the lambda parameter of $V1$ is F), combined via $\cdot$ with the one assigned in $V2$ to the same family (i.e., $F_{V1} \cdot F_{V2}$):*
{ID: "11", Name: "Calcitonin", Committee: ["Hay", "Poyner"]} $\cdot$ {ID: "11", Name: "Calcitonin", Text: "The calcitonin peptide family ", Contributors: ["Brown", "Smith"]}.

So far we have only defined the citation for a single binding. Multiple bindings lead to multiple *alternative* citations, which we capture using $+$.

DEFINITION 3.2. *Let Q, V, Q' be as in Definition 3.1, and let β_t be the set of all bindings for Q' that yield a tuple t. The citation for t w.r.t. Q, Q' (denoted as $cite(t, Q, Q', V)$) is denoted as $\Sigma_{B \in \beta_t} cite(t, Q, Q', V, B)$.*

EXAMPLE 3.2. *Recall $Q1$ and assume now that the family name N="Calcitonin" is shared by two families, with identifiers* 11 *and* 12*. This leads to two bindings to the variables of $Q1$, and intuitively to two possible ways of using the views to get the output tuple ("Calcitonin"). The citation for the tuple, in this case, will be a "+" over the expression in Example 3.1 and a similar expression for family id* 12*.*

Similarly, a query may have multiple rewritings, each leading to a possibly different citation. These are again alternatives, but the function used to combine the citations for them may be different than the one used for multiple bindings for a single rewriting. We therefore use $+^R$ ("+ for rewritings") to denote this function, whose operands are elements of the citation semiring (i.e. polynomials using $+$ and $\cdot$). $+^R$ has a neutral value 0_R, and is associative and commutative. Note that this is a formal semantics, not a means of computation: going through all rewritings would be an impractical implementation.

DEFINITION 3.3. *Let Q, V be as in Definition 3.1, and let $\mathcal{Q}$ be the set of possible rewritings of Q using V. The citation for t w.r.t. Q (denoted as $cite(t, Q, V)$) is denoted as $\Sigma^R_{Q' \in \mathcal{Q}} cite(t, Q, Q', V)$.*

EXAMPLE 3.3. *Consider the query Q from Example 2.2, and recall its two rewritings Q_1, Q_2 (assume for simplicity these are its only rewritings). Now consider* input *tuples $Family$("13", "b", "gpcr") and $FamilyIntro$("13", "$Family$b"). Together they yield an output tuple ("b") for Q. According to Q_1, the citation for ("b") should be C_{V1}(F,N,Pn)("13") $\cdot$ C_{V2}(F,Tx)("13"). According to Q_2 we get a different citation: C_{V4}(F,N,Ty)("gpcr") $\cdot$ C_{V2}(F,Tx)("13"). Combining the two citations and assuming distributivity of $\cdot$ over $+^R$, we get the final citation for the tuple:*
(C_{V1}(F,N,Pn)("13") $+^R$
C_{V4}(F,N,Ty)("gpcr")) $\cdot$ C_{V2}(F,Tx)("13").

Note that, since we sum (via $+^R$) over all possible rewritings, the citations obtained for two equivalent queries will always be the same (since by definition, equivalent queries have the same set of rewritings); in other words, the citations are insensitive to query plans, as expected.

To conclude the model, note that often one is interested in obtaining a single citation for the entire output. This is captured by an abstract *aggregation* function (Agg). We only require that it is associative and commutative and with some neutral value.

DEFINITION 3.4. *The citation for a query Q w.r.t. a set of views V and an input database D (denoted as $cite(D, Q, V)$) is defined as $Agg_{t \in Q(D)} cite(t, Q, V)$.*

Agg can also be used, through its neutral value, to include citations that are needed regardless of the query output, in particular when the output is empty. For example, this could be the database name or its NAR Database issue publication.

EXAMPLE 3.4. *Consider a rewriting Q_1 such that for every citation view V used, every lambda parameter of V is equated to a constant in Q_1 (or V has no lambda terms). In this case, all assignments to Q_1 yield exactly the same citation. Assuming that $+$ is idempotent ($a + a = a$, e.g. as in set union), we get a single citation (possibly with multiple parts, combined via $\cdot$) for each tuple. If we further assume that Agg is idempotent then we get a single citation for this rewrite* for all output tuples*.*

Since such a rewriting would lead to a very concise citation, it is plausible that it would be preferable to all other possible rewrites (see discussion of the order relation over views below). Under this preference function, we then get a single citation (multiplicand) for the entire result set*.*

3.3 Interpreting the Functions

So far we have kept the functions $+$, $+^R$, $\cdot$ and *Agg* abstract. There are several natural interpretations of these functions that the database owners could use, which we illustrate below. For example, $\cdot$ could be union, join, or the least upper bound in some ordering of the views. Likewise, $+$ could be union, and $+^R$ could *min* for some ordering on (sets of) views (see discussion of the order relation below).

EXAMPLE 3.5. *Recall the citation result*
{ID: "11", Name: "Calcitonin", Committee: ["Hay", "Poyner"]}
$\cdot$ {ID: "11", Name: "Calcitonin", Text: "The calcitonin peptide family ", Contributors: ["Brown", "Smith"]}
One natural interpretation of $\cdot$ is simply the union of the records, leading to a citation of the form:
{ {ID: "11", Name: "Calcitonin", Committee: ["Hay", "Poyner"]},
{ID: "11", Name: "Calcitonin", Text: "The calcitonin peptide family ", Contributors: ["Brown", "Smith"]} }
A different choice of $\cdot$ "joins" the records, i.e. factors out common elements, to obtain:
{ID: "11", Name: "Calcitonin", Committee: ["Hay", "Poyner"], Text: "The calcitonin peptide family ", Contributors: ["Brown", "Smith"]}
Similarly, we could interpret $+^R$ as union or join, e.g.:
{ID: "11", Name: "Calcitonin", Committee: ["Hay", "Poyner"]}
$+^R$ {ID: "11", Committee: ["Brown"], Contributors: ["Smith"]}
=
{ID: "11", Name: "Calcitonin", Committee: ["Hay", "Poyner", "Brown"]}, Contributors: ["Smith"]}
Aggregation can be used to combine the citations for individual output tuples, e.g. by using union and/or merge, in addition to adding the database name (as explained above), to form a single citation for the result set.

3.4 Introducing Order

As explained in Section 2.3, some citations are preferable to others. Such preference relations can be encoded through partial orders, as follows. We first define a partial order $\leq$ over *monomials* in the citation semiring (possibly stemming from some order relation over base annotations, see examples). We then impose that $a + b = a$ if $b \leq a$: intuitively if we can obtain two different citations and one is preferable to the other, we would like to keep only the preferable one. Such order relation can then be lifted to order relation over polynomials: to compare polynomials p_1 and p_2 we first tranform each polynomial into a "normal form", removing every monomial M_2 for which there exists a monomial $M_1 \geq M_2$. Then, we say that $p_2 \leq p_1$ if for every monomial M_2 in the normal form of p_2 there exists a monomial

M_1 in the normal form of p_1 such that $M_2 \leq M_1$. Finally, we impose $p_1 +^R p_2 = p_1$ if $p_2 \leq p_1$. Intuitively, given two possible rewritings such that a citation obtained for one is preferable to that obtained for the other, we keep only the preferable one.

We next exemplify some order relations over monomials that could be used in the above construction.

EXAMPLE 3.6. *Consider the preference relation preferring rewritings using the smallest number of views. For that, we can define $M_1 \leq M_2$ for two monomials M_1, M_2 if the number of multiplicands in M_1 is greater or equal to that of M_2 (note that we only cite views, not base relations).*

EXAMPLE 3.7. *Consider preferring rewritings using as few "uncovered" terms as possible. For that we need a slight technical modification to our model, where we designate a citation atom C_R to be placed in the citation whenever the query uses a base relation R. Now we can define $M_1 \leq M_2$ for two monomials M_1, M_2 if the number of atoms of the form C_R in M_1 is greater or equal than that in M_2.*

EXAMPLE 3.8. *Finally, consider an order relation on* views *based on view inclusion. Intuitively preferring the use of V_1 over V_2 if V_1 is included in V_2, ensures that we are citing views that are "best fit" to our needs rather than very general ones. To capture that, define $a \leq b$ if a, b are citations stemming from the citation functions for V_1, V_2 respectively, and V_2 is included in V_1. This may be lifted to monomials, e.g. we may first "normalize" each monomial w.r.t. the order relation, i.e. define $a \cdot b = a$ if $b \leq a$, and then define $a_1 \cdot \ldots \cdot a_n \leq b_1 \cdot \ldots \cdot b_n$ if for every a_i there exists a b_j such that $a_i \leq b_j$. Intuitively, for every view used in the first citation, there is a more compact view that could be used instead in the second citation (so the latter is preferable).*

With such an order relation in place, there is hope for generating a citation for a query output which avoids an exhaustive materialization of all rewritings. How to do so efficiently, and under which conditions on the views, semiring and order relation, is a fundamental task for future work.

4. CONCLUSIONS

We present an approach for automatically constructing citations to information extracted from a database via general queries. In this approach, owners of the database specify citations to a small set of (possibly parameterized) views of the database which represent typical usage patterns; they also give interpretations to the combining functions $+$, $\cdot$, $+^R$, and *Agg*. A query against the database is then rewritten using the views, and a citation for the result set constructed using the interpretations of the combining functions.

The model is robust, describing tuple-level citation annotations but also lifting citations to groups of tuples in the views, and thus indirectly to query results. It can therefore be used for constructing citations using only schema level information (as is done in query optimization using views) as well as using citation annotations attached to tuples (as is done with reasoning about provenance). Note that reasoning at the tuple level requires changes to an existing database both in terms of the schema (to capture citation annotations on view tuples) and in terms of query processing (to combine citation annotations).

A number of interesting research questions arise from this approach. First, while we have formally defined a model for citations for query results, we have not given efficient means for computing them. In particular, it is infeasible both in terms of run time and the size of the resulting citation to go through all rewritings and all assignments within each of them. A precursor for algorithms in this respect is further modeling of (some of) the "black boxes" of the model, which include the citation functions, the semiring operations, and the order relations. We have demonstrated initial ideas in this respect, such as the assumption of idempotence over $+$ and its use in some cases. Another promising direction is designing a language for the specification of the black boxes, allowing for their analysis.

Even for restricted cases, designing efficient algorithms for computing citations is a non-trivial task. To this end, our future work will also study further connections to relevant classic problems in the literature such as query inclusion (which we have mentioned as useful for the preference relation); query rewriting using views; using logs to understand database usage and decide what citation views should be specified; caching and materialization; and the maintenance and presentation of data provenance.

Last, we have not discussed *fixity*: data may evolve over time, and citations should bring back the data as seen at the time it was cited. Thus data sources must support versioning, and citations must include timestamps or version numbers. Furthermore, the choice of proper citation for output tuples may change. This may be captured in our model by including a "timestamp" attribute in base relations, with lambda variables in views corresponding to this attribute. Then, citations could vary across timestamps, and our algebraic operators may be used to aggregate (or choose some out of) these citations. Further exploring the evolution of citations is another intriguing goal for future work.

Acknowledgments. The authors would like to thank Peter Buneman and Val Tannen for many fruitful discussions related to this work. In particular, Peter Buneman initially formulated several of these ideas in the context of hierarchical databases. This work has been partially funded by NSF IIS 1302212, by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, by the Israeli Science Foundation (1636/13) and by a grant from the Blavatnik Interdisciplinary Cyber Research Center.

5. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] F. N. Afrati, C. Li, and J. D. Ullman. Using views to generate efficient evaluation plans for queries. *J. Comput. Syst. Sci.*, 73(5):703–724, 2007.

[3] P. Buneman, S. Davidson, and J. Frew. Why data citation is a computational problem. *CACM*, 59, 2016 (to appear).

[4] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[5] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[6] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[7] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[8] A. J. Pawson, J. L. Sharman, et al. The IUPHAR/BPS Guide to PHARMACOLOGY: an expert-driven knowledgebase of drug targets and their ligands. *Nucleic acids research*, 42(D1):D1098–D1106, 2014.