Provenance-directed Chase&Backchase

Alin Deutsch[*]and Richard Hull

Abstract

The Chase&Backchase algorithm for rewriting queries using views is based on constructing a canonical rewriting candidate called a universal plan (during the chase phase), then chasing its exponentially many subqueries in search for minimal rewritings (during the backchase phase). We show that the backchase phase can be sped up significantly if we instrument the standard chase to maintain provenance information. The particular provenance flavor required is known as minimal why-provenance in the literature, and it can be computed by exploiting the analogy between a chase step execution and query evaluation.

1 Chase&Backchase

The Chase&Backchase ($\mathcal{C}\&\mathcal{B}$) is an algorithm for rewriting queries using views while exploiting integrity constraints. It was introduced in [DPT99] for queries and views expressed as conjunctive queries (CQ) and integrity constraints expressed as embedded dependencies [AHV95] ([DPT99] extends the $\mathcal{C}\&\mathcal{B}$ to conjunctive queries over complex-valued and OO data models, which we do not treat here).

The $\mathcal{C}\&\mathcal{B}$ algorithm is based on expressing the view definitions as a set $\mathcal{V}$ of embedded dependencies, then chasing with these as well as with the integrity constraints $\mathcal{I}$. Let us denote the result of chasing a query Q with a set of embedded dependencies $\mathcal{D}$ as $Q^{\mathcal{D}}$. [1]

The $\mathcal{C}\&\mathcal{B}$ algorithm proceeds in two phases:

Chase: The input query Q is chased with the view constraints $\mathcal{V}$ and integrity constraints $\mathcal{I}$, to obtain a chase result $Q^{\mathcal{V}\cup\mathcal{I}}$. Next, the subquery U of $Q^{\mathcal{V}\cup\mathcal{I}}$ is produced by restricting $Q^{\mathcal{V}\cup\mathcal{I}}$ to the vocabulary of views. U is called the *universal plan* U.

[*]This author was supported by the NSF under grant IIS:1117527.

[1]We confine ourselves here to the case when the chase terminates, thus yielding a finite result. It is well-known that this result is not necessarily unique, as it depends on the non-deterministic choices made during the chase sequence among simultaenously applicable chase steps. However, the result is unique up to homomorphic equivalence [AHV95], which suffices for our purposes. We will therefore refer to "the" chase result in the remainder of this paper.

Backchase: The subqueries of the universal plan U are checked for equivalence (under $\mathcal{I}$ and $\mathcal{V}$) to Q, and all equivalent subqueries are output (as long as they are minimal, i.e. contain no subqueries that are already equivalent to Q). The equivalence check involves chasing each subquery sq "back" to Q (more precisely, checking that Q has a containment mapping into $sq^{\mathcal{V}\cup\mathcal{I}}$).

We illustrate the $\mathcal{C}\&\mathcal{B}$ algorithm on the following running example (for simplicity, without integrity constraints).

Example 1.1 *Consider the query*

$$Q(x) : -R(x,w,y), S(y,z), T(z,u)$$

and assume that the following views have been defined:

$$\begin{aligned} V_R(x,y) &\ :- \quad R(x,w,y) \\ V_S(y,z) &\ :- \quad S(y,z) \\ V_{RS}(x,z) &\ :- \quad R(x,w,y), S(y,z) \\ V_T(z,u) &\ :- \quad T(z,u) \end{aligned}$$

It is easy to see that

$$\begin{aligned} R_1(x) &\ :- \quad V_R(x,y), V_S(y,z), V_T(z,u) \\ R_2(x) &\ :- \quad V_{RS}(x,z), V_T(z,u) \end{aligned}$$

are equivalent rewritings of Q using the views. Also, each rewriting is minimal, in the sense that no atom of its definition can be removed while preserving equivalence to Q.

To find these rewritings, the $\mathcal{C}\&\mathcal{B}$ algorithm prescribes capturing the view definitions by a set $\mathcal{V}$ of embedded dependencies. These are obtained canonically by stating the inclusion (in both directions) between the result of the query defining each view and the view's extent. For the example, $\mathcal{V}$ is the following set of dependencies (as usual in the literature, free variables are to be read as universally quantified):

$$\begin{aligned} c_{V_R} : &\quad R(x,w,y) \to V_R(x,y) \\ b_{V_R} : &\quad V_R(x,y) \to \exists w\ R(x,w,y) \\[1em] c_{V_S} : &\quad S(y,z) \to V_S(y,z) \\ b_{V_S} : &\quad V_S(y,z) \to S(y,z) \\[1em] c_{V_{RS}} : &\quad R(x,w,y) \wedge S(y,z) \to V_{RS}(x,z) \\ b_{V_{RS}} : &\quad V_{RS}(x,z) \to \exists w,y\ R(x,w,y) \wedge S(y,z) \\[1em] c_{V_T} : &\quad T(z,u) \to V_T(z,u) \\ b_{V_T} : &\quad V_T(z,u) \to T(z,u) \end{aligned}$$

The chase phase. *When chasing Q with $\mathcal{V}$, the only chase steps that apply involve $c_{V_R}, c_{V_S}, c_{V_T}, c_{V_{RS}}$, yielding chase result*

$$Q^{\mathcal{V}}(x) :- R(x, w, y), S(y, z), T(z, u), V_R(x, y), V_S(y, z), V_T(z, u), V_{RS}(x, z).$$

The restriction of $Q^{\mathcal{V}}$ to the schema of the views yields the universal plan

$$U(x) :- V_R(x, y), V_S(y, z), V_T(z, u), V_{RS}(x, z).$$

The backchase phase. *In this phase, the subqueries of U are inspected. Notice that R_1, R_2 above are among them.*

We illustrate only for the suqbquery of U corresponding to R_2. To show that R_2 is equivalent to Q, we chase R_2 with $\mathcal{V}$ and we search for a containment mapping from Q into $R_2^{\mathcal{V}}$. The only applicable chase steps involve $b_{V_{RS}}, b_{V_T}$, yielding the result

$$R_2^{\mathcal{V}}(x) :- V_{RS}(x, z), V_T(z, u), R(x, w, y), S(y, z), T(z, u).$$

Since the identity mapping on variables is a containment mapping from Q to $R_2^{\mathcal{V}}$, R_2 is equivalent to Q, and thus a rewriting. R_2 is moreover minimal, since none of its subqueries is a rewriting of Q (the backchase checks this). R_2 is therefore output by the $\mathcal{C}\&\mathcal{B}$ algorithm.

R_1 is discovered analogously.

It turns out that there are no other minimal rewritings of Q. The backchase phase determines this by systematically checking the other subqueries of U, but discarding them as not being equivalent to Q, or not being minimal. For instance, the subquery

$$sq(x) :- V_R(x, u), V_T(z, u)$$

is not a rewriting of Q, and the subquery

$$sq'(x) :- V_S(y, z), V_T(z, u), V_{RS}(x, z)$$

is a rewriting but is not minimal.

Completeness of the $\mathcal{C}\&\mathcal{B}$ Algorithm. The fact that rewritings R_1 and R_2 in Example 1.1 are discovered among the subqueries of U is not accidental. In [DPT99], it was shown that all minimal rewritings of Q are (isomorphic to) subqueries of U, in the absence of integrity constraints. The result was extended to the presence of integrity constraints expressed as embedded dependencies as long as the chase with them terminates (Theorem 1 in [DT03b]; the proof can be found in [Deu02]; see also [DPT06]). The result was further extended to queries and views expressed as unions of conjunctive queries, and disjunctive embedded dependencies [DT03b, Deu02].

Implementation of $\mathcal{C}\&\mathcal{B}$ rewriting. The first $\mathcal{C}\&\mathcal{B}$ implementation is described in [PDST00], where the backchase phase is identified as the performance bottleneck. This is expected, since exponentially many subqueries of the universal plan are checked for equivalence with the original query, and each equivalence check involves a chase. While [DPT99] shows that this brute-force search is optimal from a complexity-theoretic point of view, [PDST00] concerns itself with practical feasibility and proposes techniques for pruning the search while preserving completeness. Essentially, these boil down to enumerating subqueries of the universal plan U in a bottom-up fashion, starting with all single-atom subqueries, next with two-atom subqueries, etc. Since the backchase searches for minimal rewritings, this bottom-up strategy allows pruning the equivalence check for all subqueries sq of U that already include a rewriting as subquery, since all such sq are non-minimal. In Example 1.1, subquery sq' would be pruned this way.

Even with bottom-up pruning, exponentially many subqueries remain to be chased in the worst case. In practice, this worst case occurs often, for instance when there is no rewriting of the query using the views. In this case the pruning never kicks in and all possible subqueries of U need to be checked. To deal with this case, [PDST00] proposes first checking that Q has a rewriting, before even starting the subquery enumeration. This check is performed as follows.

A corollary of the completeness of the $\mathcal{C}\&\mathcal{B}$ algorithm states that Q has a rewriting using the views if and only if it has a containment mapping into $U' = U^{\mathcal{V}\cup\mathcal{I}}$, i.e. into the result of chasing the universal plan U with the dependencies in $\mathcal{V}$ and $\mathcal{I}$. In practical implementations (e.g. in [PDST00]), the existence of a containment mapping from Q into U' is checked by treating U' as a small symbolic database instance (known as "canonical" instance in the literature [AHV95]), and evaluating Q over it. This amounts to computing the set of *all* containment mappings from Q into U', and checking its non-emptiness.

Example 1.2 *Revisiting Example 1.1, a possible chase sequence of U with $\mathcal{V}$ involves, in order, chase steps with b_{RS}, b_R, b_S and b_T, yielding*

$$\begin{aligned} U^{\mathcal{V}}(x) \quad :- \quad & V_R(x,y), V_S(y,z), V_T(z,u), V_{RS}(x,z), \\ & R(x,w_1,y), S(y,z), T(z,u), R(x,w_2,y_2), S(y_2,z). \end{aligned}$$

If we evaluate Q over the canonical instance of $U^{\mathcal{V}}$, we obtain the containment mappings $h_1 = \{x \mapsto x, y \mapsto y, w \mapsto w_1, z \mapsto z, u \mapsto u\}$ and $h_2 = \{x \mapsto x, y \mapsto y_2, w \mapsto w_2, z \mapsto z, u \mapsto u\}$. Therefore, U is a (redundant) rewriting of Q, and it makes sense to start inspecting its subqueries in search of minimal rewritings.

[PDST00], and the follow-up work in [DT03a] show that the effort of chasing U itself is in practice comparable to that of chasing any subquery of U, since the chase can be made particularly fast by implementing it as query evaluation over a canonical database of toy dimensions. The rewriting existence check is therefore shown to be well worth the effort: it bounds the overhead to an exponential fraction of the backchase runtime and yields up to exponential speedup (realized whenever there is no rewriting).

[PDST00] also presents a suite of techniques which further prune the search, when instead of all minimal rewritings one only seeks a cheapest rewriting according to a cost estimator. This setting is relevant in query optimization. It is shown how cost estimation can be interleaved with the bottom-up backchase. If the cost model satisfies reasonable assumptions like monotonicity, the resulting algorithm is shown to preserve the guarantee of finding a cheapest rewriting while pruning all subqueries whose cost exceeds the best found so far, even without chasing them to check if they are rewritings. In this paper we do not concern ourselves with cost-based pruning, focusing on enumeration of all minimal rewritings.

2 Provenance-Directed $\mathcal{C}\&\mathcal{B}$

The remainder of this paper shows that significantly more can be done to prune the search for all minimal rewritings while preserving completeness, assuming that the chase procedure used in the backchase phase is instrumented to maintain provenance information.

Intuitively, the original backchase enumerates the subqueries of the universal plan U in a bottom-up fashion and chases each of them in isolation from the others, to determine equivalence to the input query Q. This leads to *redundant chasing* of the atoms occurring in common within distinct subqueries of U. It also leads to *fruitless chasing* when the chosen subquery ends up not being equivalent to Q.

Example 2.1 *In our running example, the bottom-up backchase search prunes all strict superqueries of R_1, R_2 except U. This leads to pruning subqueries $V_R \wedge V_{RS} \wedge V_T$ and $V_S \wedge V_{RS} \wedge V_T$.*[2]

In addition, the backchase will prune those subqueries that do not contain the universal plan's head variables, as only safe rewritings are of interest (e.g. it will prune V_S, and $V_S \wedge V_T$).

However, the backchase still carries out fruitless chases of the following 7 subqueries of U

$$\begin{gathered} V_R, V_{RS}, \\ V_R \wedge V_S, V_R \wedge V_T, V_R \wedge V_{RS}, V_S \wedge V_{RS}, \\ V_R \wedge V_S \wedge V_{RS} \end{gathered}$$

only to determine that none of them are rewritings of Q.

One might wonder why the backchase won't more aggressively prune away all subqueries that don't even mention all relations mentioned by Q. In our example, this would immediately dismiss all 7 subqueries listed above. Note that this

[2]To avoid clutter, in the running example we specify universal plan subqueries by mentioning only the view names of involved atoms. This is not ambiguous since U contains no distinct atoms with the same view name. We also omit the specification of the distinguished variables, as these are in all cases (x).

aggressive pruning is unsafe in general, in the sense of compromising completeness of the backchase. Indeed, if the set of constraints includes tuple-generating dependencies (such as foreign key constraints), the minimal rewritings do not necessarily mention all relations mentioned by the query. It is easy to construct such examples: for instance, assume that the second component of S is a foreign key referencing the first component of T. Then subqueries $V_R \wedge V_S$ and V_{RS} are minimal rewritings that would be missed by the aggressive pruning.

Notice how, across rewritings R_1 and R_2, the common V_T atom is chased redundantly multiple times (once when chasing U, then again when chasing R_1, and also when chasing R_2, and in the fruitless chases of the three above-listed subqueries involving V_T).

Our aim is to minimize both fruitless and redundant chasing.

The solution we propose starts from the observation that, while chasing the universal plan U to check the existence of a rewriting, we simltaneously chase all of its subqueries, though not in isolation as in original backchase, but collectively. This collective chase will duplicate all chase steps of the isolated chases, possibly enabling strictly more chase steps that result from the interaction between simultaneous chases of the subqueries. More formally, the nature of the chase implies the following fact:

Fact 1 *The union of the results of chasing each subquery of U in isolation maps homomorphically into the result of chasing U itself.*

Fact 1 follows immediately from the fact that all chase steps that fire on an isolated subquery of U also fire on its isomorphic copy in U.

Now note that each containment mapping image i of Q in U' must have been introduced because of the presence of certain atoms in U (which induce a subquery of U). But by Fact 1 above, those subqueries of U who do not contribute to the creation of an image of Q even when chased collectively with the other subqueries, will certainly not do so when chased in isolation. Therefore, they cannot be rewritings of Q. Our goal is to dismiss these subqueries immediately, without chasing them in isolation, thus saving the effort when compared to the original backchase. As it turns out, we can do even better than that, avoiding the redundant effort across isolated chases of the remaining subqueries.

To this end, we propose a new backchase strategy that keeps track of the provenance of each atom a in U', where the *provenance* of a gives the set(s) of view atoms from U whose chasing led to the introduction of a into U'. This provenance information enables us to run Q over the canonical instance of U', identify each image i of Q into U', and trace it back to the subquery sq of U that is responsible for the creation of i during the chase of U. The search for rewriting candidates thus confines itself to the set of provenances of the images of Q into U'. This set is significantly smaller than the set of all subqueries of U. Indeed, in most cases we have encountered in practice, the backchase was exploring a large fraction of the exponentially many subqueries of U, even when there were only very few minimal rewritings.

Example 2.2 *We illustrate by revisiting Example 1.2. We show again $U^{\mathcal{V}}$, this time annotating the atoms of $U^{\mathcal{V}}$ with their provenance in terms of the view atoms in U. Since U contains no two atoms using the same view name, we drop the variables from the provenance annotation, to avoid clutter. The provenance annotations appear as superscripts.*

$$\begin{aligned} U^{\mathcal{V}}(x) \quad :- \quad & \overbrace{V_R(x,y)}^{V_R}, \overbrace{V_S(y,z)}^{V_S}, \overbrace{V_T(z,u)}^{V_T}, \overbrace{V_{RS}(x,z)}^{V_{RS}}, \\ & \overbrace{R(x,w_1,y)}^{V_R}, \overbrace{S(y,z)}^{V_S}, \overbrace{T(z,u)}^{V_T}, \overbrace{R(x,w_2,y_2)}^{V_{RS}}, \overbrace{S(y_2,z)}^{V_{RS}}. \end{aligned}$$

Note that the view atoms in $U^{\mathcal{V}}$ are annotated with themselves, as they are not introduced by chasing, but instead inherited directly from U. The R, S, T atoms in $U^{\mathcal{V}}$ are introduced by the chase, for instance the first R atom stems from chasing $V_R(x,y)$ with view dependency b_{V_R}, while the second R atom stems from chasing $V_{RS}(x,z)$ with $b_{V_{RS}}$.

Recall from Example 1.2 that Q has precisely two containment mapping images into $U^{\mathcal{V}}$: one given by h_1, comprising the T atom and the first R and S atoms, and another given by h_2, comprising the T atom and the second R and S atoms. The provenance of the first image of Q is $V_R \wedge V_S \wedge V_T$, which corresponds to rewriting R_1 in Example 1.1, while the provenance of the second image is $V_T \wedge V_{RS}$, corresponding to rewriting R_2 in the running example.

Notice how by computing the containment mappings of Q into $U^{\mathcal{V}}$ (a step that is already carried out in the original $\mathcal{C}\&\mathcal{B}$ algorithm), we immediately identify the two rewritings of Q, saving the fruitless individual chases of the subqueries listed in Example 2.1.

Also notice how, across the two remaining subqueries, R_1 and R_2, the common V_T atom is only chased once and for all when chasing U, saving the redundant chasing that would have resulted from chasing R_1 and R_2 in isolation (as prescribed by the original backchase).

2.1 Provenance-Aware Chase

We next detail the notion of provenance formula and how to instrument the chase to keep book of provenance information. We call the resulting chase *provenance-aware*. The proposed bookkeeping exploits the analogy between chase step application and query evaluation, with the chase-maintained provenance paralleling the *minimal why-provenance* flavor introduced for query evaluation in [BKT01].[3]

Intuitively, the provenance of an atom a is meant to specify the universal plan subqueries whose chase constructs atom a. This information is captured in the form of expressions obtained by starting with universal plan atoms as terms and combining them using logical conjunction and disjunction. To define

[3]This analogy is already exploited in the original $\mathcal{C}\&\mathcal{B}$ implementation, to speed up standard chase step evaluation.

the provenance of an atom in the chase result, we introduce some notation first. Given an atom a, its provenance formula is denoted as $\pi(a)$. For set/conjunction of atoms A, the provenance is the logical conjunction of the provenances of A's elements: $\pi(A) = \bigwedge_{a \in A} \pi(a)$.

We define the provenance-aware chase only for embedded dependencies corresponding to tuple-generating dependencies (tgds), i.e. dependencies in which the conclusion of the implication contains no equality atoms [AHV95]. This leaves out equality-generating dependencies (egds) which we do not treat here for simplicity sake. Notice that all dependencies in $\mathcal{V}$ are tgds, and in general tgds can express such integrity constraints as inclusion dependencies and beyond, but cannot express key constraints and functional dependencies in general.

Provenance-aware Chase Step. The provenance-aware chase of the universal plan builds provenance formulae inductively as follows:

- For each atom a of the universal plan U, let $\pi(a) = a$.
- Let ρ be an instance. Let d be a tgd of the form

  $$d: \ \mathit{premise}(\bar{x}) \rightarrow \exists \bar{y} \ \mathit{conclusion}(\bar{x}, \bar{y})$$

  where *premise* and *conclusion* are conjunctions of relational atoms and $\bar{x}$, $\bar{y}$ are vectors of variables. Let h be a homomorphism from *premise* into ρ.

  We say that the chase step of ρ with d under h *does not apply* if there is an extension $\bar{h}$ of h to a homomorphism from *conclusion* into ρ, such that $\pi(h(\mathit{premise}))$ implies $\pi(\bar{h}(\mathit{conclusion}))$.

  If the chase step does apply, then it yields ρ' obtained from ρ by adding new atoms precisely as the standard chase would, and annotating each of them with $\pi(h(\mathit{premise}))$. If ρ already contains atom a with provenance p_1, and the chase step introduces atom a with provenance p_2, this is represented in ρ' by keeping a single copy of a, with provenance $p_1 \vee p_2$.

Note that the provenance-aware chase constructs atoms just like the standard chase, but annotates them with provenance formulae, and has a more refined step applicability test. In the standard chase a step with tgd d under homomorpism h does not apply when the conclusion is already witnessed by atoms in ρ. In contrast, in the provenance-aware chase, we need to further make sure that these witness atoms of d's conclusion stem from the same view atoms whose chase yielded the image under h of d's premise.

Also note that the provenance formulae use logical conjunction and disjunction with their expected properties such as commutativity, distributivit, idempotence and absorption. This corresponds to the minimal why-provenance of [BKT01] and is a particular case of a provenance semiring [GKT07].

2.2 Provenance-directed Backchase

Once the universal plan U is provenance-aware-chased into result U', it is easy to "read off" the subqueries of U (if any) that chase into results that accommodate containment mappings from Q. These subqueries are rewritings of Q using the views.

To find them, we simply run Q over U' to compute all containment mappings from former to latter. We denote their set with $\mathcal{H}$. For each containment mapping $h \in \mathcal{H}$, the provenance information of Q's image under h, $\pi(h(Q))$, gives the subquery of U whose chase led to this image (and therefore is a rewriting of Q). Let us denote this set of rewritings as $\mathcal{R} = \{\pi(h(Q)) \mid h \in \mathcal{H}\}$. It can be shown that set $\mathcal{R}$ contains all minimal rewritings of Q using the views, but it may also contain some non-minimal rewritings. These are easily identified, as they contain as subquery some other rewriting from $\mathcal{R}$. The provenance-directed backchase purges these rewritings from $\mathcal{R}$ and returns the result.

The above processing can be equivalently cast in terms relating to querying provenance-annotated databases:

The provenance-directed backchase consists in running Q over the canonical instance of U' while keeping track of the *minimal why-provenance* [BKT01] of the result. The provenance of the tuple corresponding to Q's distinguished variables corresponds straightforwardly to subqueries of U, all of which are returned.

Example 2.3 *Recalling Example 2.2, the provenance of tuple (x) in the answer of Q over $U^{\mathcal{V}}$ is $(V_R \wedge V_S \wedge V_T) \vee (V_{RS} \wedge V_T)$, which is minimal (neither conjunct contains the other). Each of the conjuncts corresponds to a rewriting of Q: the first to R_2, the second to R_1.*

2.3 Putting It All Together

We summarize the provenance-aware $\mathcal{C}\&\mathcal{B}$ below.

```
algorithm $\mathcal{C}\&\mathcal{B}^{\mathcal{I}}_{\mathbf{V}}$
params: set $\mathbf{V}$ of CQ views, captured using set $\mathcal{V}$ of tgds,
        set $\mathcal{I}$ of integrity constraints expressed as tgds with terminating chase

input:  CQ query $Q$,
output: all minimal CQ rewritings of $Q$ using views from $\mathbf{V}$ under $\mathcal{I}$

//chase phase:
1. compute universal plan $U$
   by standard-chasing $Q$ with $\mathcal{V} \cup \mathcal{I}$ and keeping only view atoms

//provenance-directed backchase phase:
2. compute $U'$ by provenance-aware-chasing $U$ with $\mathcal{V} \cup \mathcal{I}$
3. run $Q$ over $U'$, computing the
   minimal why-provenance of the tuple corresponding to $Q$'s head variables.
4. return the subqueries of $U$ defined by this provenance.
```

We can show that the directed backchase preserves completeness:

Theorem 2.1 *If the set of integrity constraints $\mathcal{I}$ consists of tgds only (with terminating chase), then the provenance-directed $\mathcal{C}\&\mathcal{B}$ is sound and complete. That is, $\mathcal{C}\&\mathcal{B}^{\mathcal{I}}_{\mathbf{V}}$ finds all and only the minimal rewritingsof the input query using the views $\mathbf{V}$ under $\mathcal{I}$.*

3 Conclusion

Chase step execution is in essence query evaluation, and therefore there is a natural way to extend the standard chase to be provenance-aware. This exention is particulary useful when the chase is employed within the $\mathcal{C}\&\mathcal{B}$ algorithm for rewriting queries using views. By using provenance-aware chasing during the $\mathcal{C}\&\mathcal{B}$'s backchase phase, we can directly "read" the rewritings from the result of chasing back the universal plan U, thus saving the effort of running isolated chases for exponentially many subqueries of U.

Note that instrumenting the standard chase to keep provenance information introduces ovehead at runtime. We expect this overhead to be negligible, being more than made up for by the performance savings over the standard backchase. Definitive confirmation requires experimental evaluation, which we leave for future work.

Acknowledgement

This work is dedicated to Peter Buneman. Both the $\mathcal{C}\&\mathcal{B}$ project and the provenance project starting with [BKT01] originated in Penn's Database Lab. At the time, Peter was playing a key leadership role in the lab, and the first author was a graduate student educating himself on the chase by reading the chapter in [AHV95] written by the second author.

References

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[BKT01] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[Deu02] Alin Deutsch. *XML Query Reformulation Over Mixed and Redundant Storage*. PhD thesis, University of Pennsylvania, 2002.

[DPT99] Alin Deutsch, Lucian Popa, and Val Tannen. Physical data independence, constraints, and optimization with universal plans. In *VLDB*, pages 459–470, 1999.

[DPT06] Alin Deutsch, Lucian Popa, and Val Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.

[DT03a] Alin Deutsch and Val Tannen. Mars: A system for publishing xml from mixed and redundant storage. In *VLDB*, pages 201–212, 2003.

[DT03b] Alin Deutsch and Val Tannen. Reformulation of xml queries and constraints. In *ICDT*, pages 225–241, 2003.

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[PDST00] Lucian Popa, Alin Deutsch, Arnaud Sahuguet, and Val Tannen. A chase too far? In *ACM SIGMOD Conference*, pages 273–284, 2000.