Query Reformulation with Constraints*

Alin Deutsch
University of California at San Diego
deutsch@cs.ucsd.edu

Lucian Popa
IBM Almaden Research Center
lucian@almaden.ibm.com

Val Tannen
University of Pennsylvania
val@cis.upenn.edu

1 Introduction

Let Σ_1, Σ_2 be two schemas, which may overlap, $\mathcal{C}$ be a set of constraints on the joint schema $\Sigma_1 \cup \Sigma_2$, and q_1 be a Σ_1-query. An **(equivalent) reformulation** of q_1 in the presence of $\mathcal{C}$ is a Σ_2-query, q_2, such that q_2 gives the same answers as q_1 on any $\Sigma_1 \cup \Sigma_2$-database instance that satisfies $\mathcal{C}$. In general, there may exist multiple such reformulations and choosing among them may require, for example, a cost model.

In 1999 we published an algorithm, called Chase and Backchase (C&B), for enumerating the reformulations of a query under constraints [11]. Our main motivation was query optimization, in which Σ_1's role is played by the *logical schema* and Σ_2's role by the *physical schema*. We found that the assertions used for integrity constraints (a.k.a. dependencies), by relating the elements of the logical and physical schemas constitute a flexible tool for modeling ideas such as "semantic" optimization [4], and the use of cached data or materialized views [33, 3].

The 1999 paper did not limit itself to the standard relational model and instead, following [30] and more distantly [6, 23], covered complex values and OO classes with extents. A comprehensive approach to query optimization for this model, including join (usual and dependent) reordering, appeared in [28], see also[29].

Query reformulation is also essential for data publishing [32, 12] where Σ_1 is the public schema and Σ_2 the proprietary (storage) schema. It is equally essential in schema evolution where Σ_1 respectively Σ_2 is the old, respectively new schema.

Since views can be modeled as a pair of inclusion constraints, the C&B algorithm provided a new technique for rewriting with views [25] and hence was also applicable to information integration. In fact, we had already shown in [11] that C&B will find all reformulations of conjunctive queries using conjunctive views, if such reformulations exists. However, we should emphasize that C&B finds *equivalent* reformulations while in information integration, when equivalent reformulations may not exist, one is also very much interested in reformulations that produce some (as many as possible) of the answer tuples [26, 1, 21, 7].

As its name suggest, C&B is using the *chase*, a technique developed 25+ years ago for the purposes of deciding logical consequence for most types of integrity constraints used in databases [27, 5]. Many papers have used the chase since then [1] . It seemed surprising that there would still exist fundamental properties of the chase left undiscovered. Nonetheless, we thought that the C&B algorithm provided such a property. This was formally verified in [13] where we proved that with constraints to which the chase applies, whenever the chase terminated, C&B would find all minimal reformulations of conjunctive (select-project-join) queries.

This completeness property holds also for the complex values and OO model, using a generalization of the chase developed in [30]. Moreover, the C&B algorithm was used also for the reformulation of XML queries, via a compilation from XML to relational queries and constraints [14, 12, 10]. These early successes encourage us to think that C&B could become a versatile tool for query processing. This survey will attempt to provide an introduction to the why, when, and especially how, of C&B.

2 What is C&B?

From the beginning it was observed that the chase can also be used to decide containment (hence equivalence) of conjunctive queries in the presence of constraints. Indeed, if the chase of q_1 with $\mathcal{C}$ terminates producing a query q_c then $q_1 \subseteq_{\mathcal{C}} q_2$ iff $q_c \subseteq q_2$ and the latter can be checked by finding a containment mapping from q_2 to q_c [9, 2]. (Here, $q_1 \subseteq_{\mathcal{C}} q_2$ means that when q_1 and q_2 are applied to any instance that satisfies $\mathcal{C}$ the answers of q_1 are contained in those of q_2. Similarly for $\equiv_{\mathcal{C}}$.)

In the reformulation problem, however, we are only given $\mathcal{C}$ and q_1 and we must decide whether there exists a q_2 such that $q_1 \equiv_{\mathcal{C}} q_2$. Since q_2 is among infinitely many queries of the same type as q_1 deciding this isn't obvious. Moreover, in practice we want to actually compute a q_2 when it exists, in fact we probably want to enumerate the q_2's that provide solutions and choose among them based on cost criteria. But it's easy to see that queries can be syntactically "padded" with redundant joins while conserving equivalence, ad infinitum. We are therefore led to searching for solutions that satisfy some syntactically determined *minimality* condition. (See the definition of minimality under constraints in section 4.) As a consequence, we shall

*Database Principles Column. Column editor: Leonid Libkin, Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3H5, Canada. E-mail: libkin@cs.toronto.edu.

[1] In this survey we assume familiarity with conjunctive queries, homomorphisms and the chase procedure which are all covered extensively in [2]. To keep the paper self-contained we review these definitions in the appendix.

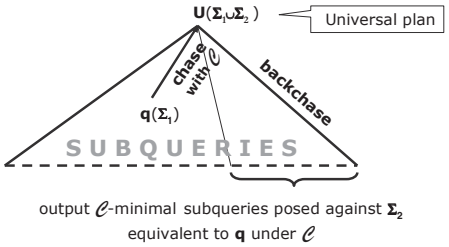

Figure 1: Chase and Backchase.

solve both the reformulation problem and a generalization of the query minimization problem [9, 2].

The C&B algorithm applies to the case when q_1 is a *conjunctive query* and when the constraints in $\mathcal{C}$ are either a *tuple-generating dependency (tgd)* of the form

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$$

or an *equality-generating dependency (egd)* of the form

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow (x_1 = x_2))$$

(see [5]). Here, $\phi(\mathbf{x})$ and $\psi(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over $\Sigma_1 \cup \Sigma_2$, all of the variables in $\mathbf{x}$ must appear in $\phi(\mathbf{x})$, and x_1, x_2 must be among the variables in $\mathbf{x}$.

These two classes (tgds and egds) together comprise the (embedded) implicational dependencies [16], which seem to include essentially all of the naturally-occurring constraints on relational databases. Furthermore, tgds, which were originally meant as a generalization of integrity constraints such as join and inclusion dependencies turn out to be ideally suited for describing schema mappings in data exchange [18] and data integration [24], as well as for capturing physical structures typically used in query optimization (views, indexes, join indexes, gmaps, etc.) [11]. As a whole, the class of embedded implicational dependencies is remarkably well-suited for representing most intra- and inter-schema relationships that are of importance in practice.

C&B proceeds in two phases. In the **chase phase** it uses $\mathcal{C}$ to chase q_1 until (and if) no more chase steps are possible. We call the resulting query U, a **universal plan**, see Figure 1

Now it's time to recall that q_1 is Σ_1-query, that we are looking for a Σ_2-query as reformulation, and that the constraints in $\mathcal{C}$ are on the joint schema $\Sigma_1 \cup \Sigma_2$. The universal plan, U, resulting from the chase of q_1 with $\mathcal{C}$ will (in general) be a $\Sigma_1 \cup \Sigma_2$-query. We can think of the universal plan as incorporating *all* possible alternative ways to answer q_1 in the presence of the constraints $\mathcal{C}$. This intuition is fully justified by the following [13]:

Theorem 1 *If q_m is a minimal conjunctive $\Sigma_1 \cup \Sigma_2$-query equivalent to q_1 under $\mathcal{C}$, i.e., $q_1 \equiv_{\mathcal{C}} q_m$, then q_m is (isomorphic to) a* subquery *of the universal plan U.*

It is now possible to effectively enumerate all minimal reformulations. Indeed, we need only search the finite space of subqueries of U. This is done in the **backchase phase**, so called because we check for equivalence with q_1 by chasing subqueries of U with $\mathcal{C}$. These chase sequences go "backwards", toward the U we already have. For each such candidate reformulation we can stop (equivalence holds) whenever we have a containment mapping from U into an intermediate chase result or (no equivalence) when the chase terminates without such a containment mapping. In fact, as we shall see (Section 4), it is enough to check the existence of a containment mapping from the original query q_1 into any intermediate result of chasing the candidate subquery of U.

We see that in both the chase and the backchase phase the algorithm (and Theorem 1) needs the chase sequences to terminate. In [5] it was shown that this is always the case if the tgds are *total* or *full* [2] (they cannot have $\exists$) while the egds can be arbitrary. While full tgds cannot in general model the physical structures or the integration/exchange mappings we have become interested in, Deutsch and Popa have recently discovered a significantly larger and remarkably useful class of tgds that can. Chase sequences with such sets of dependencies, called *weakly acyclic* in [17, 18] and *stratified-witness* in [13, 14] are guaranteed to terminate. The set of constraints from Example 2 in Section 3 is weakly acyclic.

Finally, note that the subqueries of U are in general $\Sigma_1 \cup \Sigma_2$-queries. Some of them may in fact be Σ_1-queries (q_1 itself is one!) and some may be Σ_2-queries. The theorem above guarantees that if Σ_2-reformulations exist, then we shall find all minimal ones among the subqueries of U.

3 Schemas and Constraints, Queries and Rewritings

In this article we focus our presentation on a scenario where query reformulation is applied to a distributed heterogeneous environment, with multiple schemas that are interconnected by complex relationships. The problem is that of finding alternative (and equivalent) reformulations of a query that is initially formulated in terms of one of the schemas. Our running example will show the challenges (and opportunities) for query reformulation in such an environment. The example will depict constraints that fall into one of four categories:

1. **(Traditional) single-database constraints** (e.g., key and foreign key constraints.)
2. **Relationships (mappings) between schemas.** These constraints are a consequence of how these repositories have been created and subsequently maintained.
3. **Domain knowledge constraints.** These constraints are assertions that are true about a specific situation, for example, the fact that a customer id has a unique nation code across repositories.
4. **Constraints capturing materialized views.** These constraints express the fact that data is redundantly stored in both base tables and materialized views.

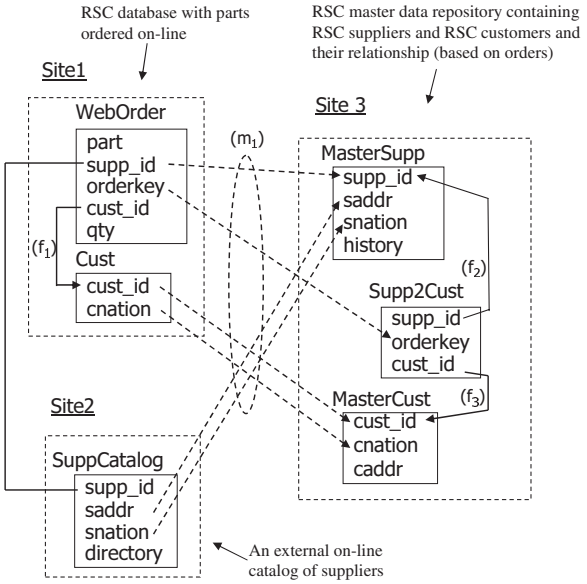

Figure 2: Retail Store Chain Example

Example 1 (Running) Consider a large retail store chain (call it RSC) maintaining and accessing several repositories with data about its suppliers, customers and parts.

One of the repositories (located at Site 2) is an external, read-only, on-line directory of suppliers. The other repositories are internal but distributed across Sites 1 and 3, with different structure, and with different although possibly overlapping data. The repository at Site 1 is a database containing parts ordered on-line and some of the associated customer and supplier information. Additional repositories like this may exist (not shown here for simplicity). The repository at Site 3 is a central repository intended to contain all the information about RSC suppliers, customers and the orders that relate them. Figure 2 illustrates the schemas of these repositories; it also depicts some of the intra- and inter-schema constraints that hold.

Example 2 (Constraints) We illustrate next some of the constraints associated with the schemas in the running example. These constraints fall under the first three categories mentioned earlier. We shall illustrate constraints in the fourth category, describing views, later in Section 5.

1. (Traditional) single-database constraints. The following egds can be used to express that $cust\_id$ plays the role of a key in the each of the tables $Cust$ and $MasterCust$ and similarly, $supp\_id$ is a key in $MasterSupp$. (As a notational convenience, we will drop the the universal quantifiers in front of a dependency, and implicitly assume such quantification.)

$$k_1: \quad Cust(c, cn) \wedge Cust(c, cn') \rightarrow cn = cn'$$
$$k_2: \quad MasterCust(c, cn, ca) \wedge MasterCust(c, cn', ca') \rightarrow (cn = cn') \wedge (ca = ca')$$
$$k_3: \quad MasterSupp(s, sa, sn, h) \wedge MasterSupp(s, sa', sn', h') \rightarrow (sa = sa') \wedge (sn = sn') \wedge (h = h')$$

The following tgds describe formally the foreign key constraints f_1, f_2, and f_3 shown in Figure 2.

$$f_1: \quad WebOrder(p, s, o, c, q) \rightarrow \exists cn\ Cust(c, cn)$$
$$f_2: \quad Supp2Cust(s, o, c) \rightarrow \exists sa \exists sn \exists h\ MasterSupp(s, sa, sn, h)$$
$$f_3: \quad Supp2Cust(s, o, c) \rightarrow \exists cn \exists ca\ MasterCust(c, cn, ca)$$

2. Relationships (mappings) between schemas. The mapping m_1 from Sites 1 and 2 to Site 3 reflects the fact that the master data repository will be refreshed with data from Site 1 and Site 2, for instance due to a periodic process that takes customer and supplier info from Site 1, joins with Site 2 to get extra supplier information (e.g., $saddr$ and $snation$) and updates appropriate tables of Site 3. Such a mapping can be specified using schema mapping tools (e.g., Clio [31]). In Figure 2, the mapping is shown informally via the dotted arrows grouped under m_1. The link between $supp\_id$ in Site 1 and $supp\_id$ in Site 2 reflects the join. Formally, the meaning of mapping m_1 is expressed by the following tgd (universal quantifiers are again dropped):

$$\begin{aligned} m_1: \quad & WebOrder(p, s, o, c, q) \wedge Cust(c, cn) \\ & \wedge\ SuppCatalog(s, sa, sn, d) \\ & \rightarrow \exists h \exists ca\ (MasterSupp(s, sa, sn, h) \\ & \wedge\ Supp2Cust(s, o, c) \wedge MasterCust(c, cn, ca)) \end{aligned}$$

Another example of a mapping between schemas (not shown in Figure 2 to avoid cluttering) is the following tgd, expressing that $SuppCatalog$ is an "authority" for supplier information, and every supplier in $MasterSupp$ at Site 3 can be found in $SuppCatalog$ at Site 2. (The converse may not be true.)

$$m_2: MasterSupp(s, sa, sn, h) \rightarrow \exists d\ SuppCatalog(s, sa, sn, d)$$

3. Domain knowledge constraints. The fact that a customer id has a unique nation code (across all repositories) is expressed by adding the following egd to the earlier key constraints:

$$e: Cust(c, cn) \wedge MasterCust(c, cn', ca) \rightarrow cn = cn'$$

Note that e is more general than a functional dependency, as it states a property about tuples in different tables.

Example 3 (Reformulations) Consider the following query (expressed in conjunctive query notation [2]):

$$\begin{aligned} q(p, c, sa, sn) \quad :- \quad & WebOrder(p, s, o, c, q), \\ & SuppCatalog(s, sa, sn, d) \end{aligned}$$

The query q retrieves all parts that were ordered at Site 1, with the addresses and nations of suppliers and with the customer ids. The query needs to access Site 1 and Site 2, to be executed in its current form.

Given the overall configuration, q is equivalent to the following (non-obvious) rewriting:

$$\begin{aligned} q'(p, c, sa, sn) \quad :- \quad & WebOrder(p, s, o, c, q), \\ & MasterSupp(s, sa, sn, h) \end{aligned}$$

The query q' accesses Site 1 and Site 3 (all within RSC) and avoids the external catalog (which could be slower, less available, may require subscription, etc). Thus, q' is potentially more efficient with respect to execution time or cost.

If for Example 1 we have that Σ_1 contains the union of the schemas at all sites and $\Sigma_1 = \Sigma_2$, then Example 3 shows that we need to consider at least two candidates for evaluation: q' and q itself. As the configuration of the system grows larger (e.g., additional databases, cached queries, materialized views, etc.), the number of equivalent rewritings increases as well (as we shall also see in a later example). This increases the potential for improvement in performance but at the same time poses the challenge of finding such reformulations in a systematic and complete way.

Section 4 describes how the C&B algorithm can be used for systematic enumeration of available reformulations. This enumeration is based on constraints such as the ones described above. In Section 5 we modify the running example by adding materialized views (one in the example). We then describe how the same C&B algorithm is able to find extra, view-based reformulations, by using additional constraints describing the views.

4 The C&B Algorithm

Given a conjunctive query q and a set $\mathcal{C}$ of constraints, the C&B algorithm finds reformulations q' of q under $\mathcal{C}$ (i.e. $q' \equiv_{\mathcal{C}} q$) which are *minimal under* $\mathcal{C}$ (in short $\mathcal{C}$*-minimal*). The notion of minimality of a conjunctive query in the absence of constraints is well-known [9, 2]: q is minimal if dropping any of its atoms compromises equivalence to q. For minimality under constraints, we require a subtle modification:

Definition 1 (Minimality under constraints) *A conjunctive query q is $\mathcal{C}$-minimal if there are no queries s_1, s_2 where s_1 is obtained from q by replacing zero or more variables with other variables of q, and s_2 by dropping at least one atom from s_1 such that s_2 and s_1 remain equivalent to q: $q \equiv_{\mathcal{C}} s_1 \equiv_{\mathcal{C}} s_2$.*

Intuitively, the variable replacement reflects the equalities between replaced and replacing variables as implied by the equality-generating dependencies (egds) in $\mathcal{C}$.

Example 4

$$\begin{aligned} q_{nm}(cn, cn') &:- Cust(c, cn), MasterCust(c, cn, ca), \\ &\quad Cust(c, cn'), MasterCust(c, cn', ca') \\ s_1(cn, cn) &:- Cust(c, cn), MasterCust(c, cn, ca), \\ &\quad MasterCust(c, cn, ca') \\ s_2(cn, cn) &:- Cust(c, cn), MasterCust(c, cn, ca) \end{aligned}$$

The query q_{nm} above yields pairs of nations of customers listed with the same customer id. The query is minimal in the absence of constraints: we cannot drop any atom, as proven by the absence of containment mappings. However, for constraint e from Example 2, q_{nm} is *not* $\{e\}$-minimal, as witnessed by queries s_1 and s_2 above. Intuitively, replacing cn' with cn preserves $\{e\}$-equivalence of s_1 to q_{nm}, since $cn = cn'$ is implied by e (the duplicate atom *Cust(c,cn)* is removed). It is easy to check that the removal of the second *MasterCust* atom preserves equivalence to s_1 even in the absence of constraints.

As illustrated in Figure 1, the C&B algorithm proceeds in two phases. In the **chase phase**, the original query q is chased with the constraints in $\mathcal{C}$, yielding the query U called a *universal plan*. The **backchase phase** enumerates all $\mathcal{C}$-minimal subqueries sq of U which are formulated against Σ_2 and are $\mathcal{C}$-equivalent to q ($sq \equiv_{\mathcal{C}} q$). (A *subquery* is obtained by dropping one or more atoms in the original query with the condition that the head variables continue to appear in the new query body.)

Example 5 (Chase) Recall the query q from Example 3:

$$\begin{aligned} q(p, c, sa, sn) &:- WebOrder(p, s, o, c, q), \\ &\quad SuppCatalog(s, sa, sn, d) \end{aligned}$$

A chase step of q with f_1 yields

$$\begin{aligned} q_1(p, c, sa, sn) &:- WebOrder(p, s, o, c, q), Cust(c, cn), \\ &\quad SuppCatalog(s, sa, sn, d) \end{aligned}$$

which chases with m_1 to

$$\begin{aligned} U(p, c, sa, sn) &:- WebOrder(p, s, o, c, q), Cust(c, cn), \\ &\quad SuppCatalog(s, sa, sn, d), \\ &\quad MasterSupp(s, sa, sn, h), \\ &\quad Supp2Cust(s, o, c), \\ &\quad MasterCust(c, cn, ca) \end{aligned}$$

which is the universal plan since no further chase step applies.

For each subquery sq of U, to check $sq \equiv_{\mathcal{C}} q$ it suffices to check $sq \subseteq_{\mathcal{C}} q$. Indeed, by construction U is contained in sq, $U \subseteq sq$, and $U \equiv_{\mathcal{C}} q$ because the chase preserves equivalence under constraints [2]. Checking $sq \subseteq_{\mathcal{C}} q$ reduces according to classical results to finding a containment mapping from q into the result of chasing sq with $\mathcal{C}$ [2]. Finding a containment mapping into an *intermediate* result of chasing also suffices to show containment.

Pruning Property. An immediate yet naive backchase implementation would exhaustively enumerate all subqueries. We can however avoid this by using the following key observation (called the *pruning property*) which follows from the definition of $\mathcal{C}$-minimality: given a subquery sq of U with $sq \equiv_{\mathcal{C}} q$, every subquery sq' of U corresponding to a superset of sq's atoms (we say that sq' is a *superquery* of sq) cannot be both $\mathcal{C}$-equivalent to q and $\mathcal{C}$-minimal.

The pruning property enables an efficient backchase implementation which enumerates subqueries of U *bottom-up*, starting with all subqueries generated by one atom of U, continuing with those generated by all pairs of atoms, then all triplets, and so on. As soon as a subquery is $\mathcal{C}$-equivalent to q, it is output and all its superqueries are pruned from subsequent consideration. This enumeration discipline avoids even generating non-minimal reformulations. We will discuss alternate backchase implementations shortly.

Example 6 (Backchase) The rewriting q' of q from Example 3 corresponds to the subquery of U generated by the *WebOrder* and *MasterSupp* atoms. Since q' has no smaller subquery, it is one of the starting points in the bottom-up backchase. To check $q' \subseteq_{\mathcal{C}} q$, we chase q' with $\mathcal{C}$. A first chase step with constraint m_2 yields

$$\begin{aligned} q_1'(p, c, sa, sn) \quad :- \quad & WebOrder(p, s, o, c, q), \\ & MasterSupp(s, sa, sn, h), \\ & SuppCatalog(s, sa, sn, d) \end{aligned}$$

Although we could continue to chase with f_1 and then m_1, it is not necessary. We can already find a containment mapping from q to q_1' (the identity mapping), thus proving that q' is equivalent under $\mathcal{C}$ to q. It can be checked that q' is $\mathcal{C}$-minimal. In fact, it is a property of the C&B algorithm that it outputs only $\mathcal{C}$-minimal reformulations.

The backchase will enumerate additional subqueries and will possibly output other minimal reformulations. For instance, the subquery of U generated by its *WebOrder* and *SuppCatalog* atoms is q itself.

In the example, the retrieval of rewriting q' as a subquery of the universal plan is not merely a happy coincidence: by Theorem 1, we have that whenever the chase is guaranteed to terminate, all minimal reformulations of a query will be found by the C&B algorithm:

Theorem 2 (Sound and complete C&B [11, 13]) *Let q be a conjunctive query and $\mathcal{C}$ a set of tgds and egds such that the chase of any query with $\mathcal{C}$ terminates. Then the C&B algorithm outputs precisely all $\mathcal{C}$-minimal reformulations of q (up to isomorphism).*

The C&B algorithm relies on the termination of the chase. This property is in general undecidable for conjunctive queries and constraints given by tgds and egds. However, the notion of *weak acyclicity* of a set of constraints, is sufficient to guarantee that any chase sequence terminates. This is the least restrictive sufficient termination condition we are aware of, holding in all scenarios we encountered in practice (including our example).

Definition 2 (Weakly acyclic set of constraints) Let $\mathcal{C}$ be a set of tgds over a fixed schema. Construct a directed graph, called the *dependency graph*, as follows: (1) there is a node for every pair (R, A) with R a relation symbol of the schema and A an attribute of R; call such pair (R, A) a *position*; (2) add edges as follows: for every tgd $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in $\mathcal{C}$ and for every x in $\mathbf{x}$ that **occurs** in ψ:

- For every occurrence of x in ϕ in position (R, A_i):
  (a) for every occurrence of x in ψ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$.
  (b) in addition, for every existentially quantified variable y and for every occurrence of y in ψ in position (T, C_k), add a *special edge* $(R, A_i) \stackrel{*}{\rightarrow} (T, C_k)$.

Note that there may be two edges in the same direction between two nodes, if exactly one of the two edges is special. Then $\mathcal{C}$ is *weakly acyclic* if the dependency graph has no cycle going through a special edge. We say that a set of tgds and egds is weakly acyclic if the set of all its tgds is weakly acyclic.

Theorem 3 ([17, 13]) *If $\mathcal{C}$ is a weakly acyclic set of tgds and egds, then the chase with $\mathcal{C}$ of any conjunctive query q terminates.*

By Theorems 3 and 2, the C&B algorithm is sound and complete for weakly acyclic sets of constraints:

Corollary 1 *If $\mathcal{C}$ is a weakly acyclic set of tgds and egds, then the C&B algorithm outputs precisely the $\mathcal{C}$-minimal reformulations of its input query.*

The complexity of the chase. For fixed schemas and set $\mathcal{C}$ of constraints, if $\mathcal{C}$ is weakly acyclic then any chase sequence terminates in polynomial time in the size of the query being chased (as shown in [17, 13]). The fixed size assumption about schemas and constraints is often justified in practice, where one is usually interested in repeatedly reformulating incoming queries for the same setting with schemas and constraints. Nonetheless, the degree of the polynomial depends on the size of the dependencies and care is needed to implement the chase efficiently. Successive implementations have shown that in practical situations the chase is eminently usable [29, 28, 10, 12].

The complexity of reformulation. Assume that the chase of any query with $\mathcal{C}$ terminates in polynomial time. Then checking whether a conjunctive query q admits a reformulation is NP-complete in the size of q. Checking whether a given query r is a $\mathcal{C}$-minimal reformulation of q is NP-complete in the sizes of q and r. Note that for arbitrary sets of dependencies (for which the chase may not even terminate), the above problems are undecidable.

Alternative strategies for backchase. The complexity of the backchase is an even more delicate issue since even though the size of the universal plan is (with weakly acyclic dependencies) polynomial in that of the original query, in the worst case the backchase enumerates exponentially many minimal solutions [29]. Above, we presented the backchase as a bottom-up procedure that generates subqueries of the universal plan U by starting from the smallest subqueries and extending them with additional atoms from U. The algorithm stops extending a subquery sq as soon as sq becomes *equivalent* to the original query q. Such sq is guaranteed to be a $\mathcal{C}$-minimal reformulation of q.

Symmetrically, another way of implementing the backchase minimization is a *top-down*, decremental, procedure that goes from the universal plan down to its subqueries by eliminating one relational atom at a time in a systematic way, starting at every step a new branch per available atom. The algorithm stops descending on a branch whenever a *non-equivalent* subquery is found. The last equivalent query on that branch is a $\mathcal{C}$-minimal reformulation.

The top-down and the bottom-up algorithms are dual and produce the exact same output. However, the bottom-up ap-

proach has a crucial advantage in that it can be mixed with *cost-based pruning* (when cost information is available). The resulting procedure is as follows. When we find the first minimal reformulation, we estimate its cost (for example, based on traditional methods that include join reordering). This cost becomes the *best cost* so far and it will be subsequently replaced by the cost of every minimal reformulation that we may find later. Once the best cost is in place, for every explored subquery, even before checking for equivalence, we compute its cost. If the cost is higher than the best cost so far, then we can prune the subquery *together with all of its superqueries* without checking equivalence. This pruning still guarantees that the least-cost reformulation is found under the (typically true[2]) assumption that queries become more expensive by adding extra atoms (joins). The improvement in performance of the overall method is then substantial, sometimes over an order of magnitude [28]. Bottom-up backchase minimization with cost-based pruning is further extended in [28] to deal also with cases in which the above assumption may be violated (for example, due to the presence of indexes). Additional exploration strategies for subqueries of the universal plan are investigated in [29]. There it was shown that in various practical situations, the C&B method can be *stratified*, which means essentially, that the universal plan can be decomposed into independent fragments (smaller universal plans). For each fragment the backchase minimization is applied in the usual way. The minimal reformulations that result for each fragment can then be put together, by joining, as minimal reformulations for the entire process. The net effect is a significant reduction in the exponent of the search space, and hence considerable improvement in the performance of the method.

5 Adding Views

We show next that materialized views defined by conjunctive queries can be captured using tgds, and hence the C&B algorithm serves in particular as a complete algorithm which finds all minimal rewritings of a conjunctive query using conjunctive query views under integrity constraints. All we need to do is add the constraints (tgds) capturing the views, and reformulate the query against a schema containing the view names.

In detail, let $\mathcal{V}$ be a set of views defined by conjunctive queries against Σ_1. The views define a relationship between schemas Σ_1 and the schema $\mathcal{V}$, in which each view name V denotes the table with the materialized result of the homonymous view. We express this relationship equivalently using the set of dependencies $\mathcal{C}_\mathcal{V}$ constructed as follows. For each view $V \in \mathcal{V}$, assume w.l.o.g. that it is defined by the query

$$V(\mathbf{x}) :- body(\mathbf{x}, \mathbf{y})$$

where *body* is a conjunctive query body and $\mathbf{x}, \mathbf{y}$ are its variables. Let d_V^1, d_V^2 be the dependencies (over schema $\Sigma_1 \cup \mathcal{V}$):

$$d_V^1 :\ body(\mathbf{x}, \mathbf{y}) \rightarrow V(\mathbf{x}) \qquad d_V^2 :\ V(\mathbf{x}) \rightarrow \exists \mathbf{y}\ body(\mathbf{x}, \mathbf{y})$$

[2] And in fact, the very idea of minimization is based on such assumption.

which state the inclusions between the result of the query defining the view, and the materialized table V. Set

$$\mathcal{C}_\mathcal{V} := \{d_V^1 \mid V \in \mathcal{V}\} \cup \{d_V^2 \mid V \in \mathcal{V}\}.$$

We consider two flavors of rewriting using views: rewritings using exclusively the views (also called *total* rewritings in [25]), and rewritings using both the views and the base tables in Σ_1 (called *partial* rewritings in [25]). Thus, given conjunctive Σ_1-query q and set of constraints $\mathcal{C}$ over Σ_1, the problem of finding all total conjunctive query rewritings of q reduces to finding all minimal reformulations of q against schema $\Sigma_2 := \mathcal{V}$ under constraints $\mathcal{C} \cup \mathcal{C}_\mathcal{V}$. For partial rewritings, we set $\Sigma_2 := \Sigma_1 \cup \mathcal{V}$. According to Theorem 2, both flavors are completely solved by the C&B algorithm.

Example 7 Continuing our example, consider the following query launched at Site 1, which retrieves all parts provided by Japanese suppliers and ordered by US customers.

$$\begin{aligned} j2us(p) \quad :- \quad & WebOrder(p, s, o, c, q), Cust(c, \text{“}US\text{”}), \\ & SuppCatalog(s, sa, \text{“}Japan\text{”}, d) \end{aligned}$$

In general, the query would need to access Sites 1 and 2. Assume however that the previously answered query q from Example 3 is cached at Site 1, in cache entry $cache_q$. Then $j2us$ has a partial rewriting which reuses the pre-computed join of $WebOrder$ and $SuppCatalog$, performing only the remaining join between the cache entry and the customer table, both located at Site 1:

$$j2us'(p) :- cache_q(p, c, sa, \text{“}Japan\text{”}), Cust(c, \text{“}US\text{”})$$

This is more efficient as it avoids network access to Site 2, and saves the time to recompute the join of $WebOrder$ and $SuppCatalog$.

The C&B algorithm discovers this rewriting when called with $\Sigma_2 := \Sigma_1 \cup \mathcal{V}$ and $\mathcal{C} \cup \mathcal{C}_\mathcal{V}$, where $\mathcal{V}$ contains the names of all active cache entries, Σ_1 is the union of the schemas at all sites, and $\mathcal{C}_\mathcal{V}$ is constructed as described above. For instance, entry $cache_q$ can be seen as the materialized view

$$\begin{aligned} cache_q(p, c, sa, sn) \quad :- \quad & WebOrder(p, s, o, c, q), \\ & SuppCatalog(s, sa, sn, d) \end{aligned}$$

and $\mathcal{C}_\mathcal{V}$ includes the constraints:

$$\begin{aligned} d^1_{cache_q} &: WebOrder(p, s, o, c, q) \wedge SuppCatalog(s, sa, sn, d) \\ &\qquad \rightarrow cache_q(p, c, sa, sn) \\ d^2_{cache_q} &: cache_q(p, c, sa, sn) \rightarrow \exists s \exists o \exists q \exists d \\ &\qquad WebOrder(p, s, o, c, q) \\ &\qquad \wedge SuppCatalog(s, sa, sn, d) \end{aligned}$$

Now $j2us$ chases with m_1, then $d^1_{cache_q}$, to

$$\begin{aligned} j2us_1(p) \quad :- \quad & WebOrder(p, s, o, c, q), Cust(c, \text{“}US\text{”}), \\ & SuppCatalog(s, sa, \text{“}Japan\text{”}, d), \\ & MasterSupp(s, sa, \text{“}Japan\text{”}, h), \\ & Supp2Cust(s, o, c), \\ & MasterCust(c, \text{“}US\text{”}, ca), \\ & cache_q(p, c, sa, \text{“}Japan\text{”}) \end{aligned}$$

This is the universal plan, and it contains the equivalent (and minimal) rewriting $j2us'$ as the subquery given by the second and last atoms. In fact, the universal plan includes even more. The following two subqueries of the universal plan are also $\mathcal{C}$-minimal reformulations of $j2us$:

$$\begin{aligned} j2us''(p) \quad :- \quad & WebOrder(p, s, o, c, q), Cust(c, \text{“}US\text{”}), \\ & MasterSupp(s, sa, \text{“}Japan\text{”}, h) \\ j2us'''(p) \quad :- \quad & WebOrder(p, s, o, c, q), \\ & MasterSupp(s, sa, \text{“}Japan\text{”}, h), \\ & MasterCust(c, \text{“}US\text{”}, ca) \end{aligned}$$

While the first of the two rewritings above is similar to the earlier rewriting q' (Example 3), the second of the two rewritings is slightly different and less obvious. Its equivalence to the original query (which can be proven by chasing) depends essentially on the existence of several of the constraints in the system (specifically, m_2, f_1, and even the egd e). The last two reformulations do not include the view ($cache_q$) but they can be equally good candidates for execution.

This example shows the versatility of the C&B method as a rewriting tool that unifies several different concepts (e.g., views, constraints, mappings) under one umbrella, that of rewriting under constraints.

6 Other Considerations

Dictionaries. An interesting property of the query and dependency languages used in [30, 11] is the use of *dictionary* structures. In conjunction with complex values, dictionaries can be used (see [11]) to model OO classes with extents, primary and secondary indexes on either relations or class extents and gmaps [33]. On one hand this allows one to express and optimize arbitrary OQL queries [8]. On the other hand, the explicit presence of indexes allows an optimizer that uses C&B to automatically discover non-trivial execution plans that would not be found by traditional optimizers (including the ones that perform rewriting using materialized views [21]).

More expressive queries and constraints. For simplicity, we have presented the C&B only for conjunctive queries and constraints given by tgds and egds. However, the soundness and completeness of the C&B carries over to unions of conjunctive queries with inequalities, and tgds and egds extended with disjunction and inequalities (using an appropriate generalization of the chase) [10, 13, 14].

XML query reformulation. We were able to apply the C&B method to XML query reformulation, by using a relational encoding of queries, views and constraints that are originally written against a schema which models the XML tree. Relationships between XML elements (such as parent-child and ancestor-descendant) are captured by relational tables satisfying certain constraints (e.g. each child has at most one parent, the descendant table is transitive, etc.). We could show that for a significant class of XML queries, the minimal reformulations are found by running the C&B algorithm on the relational encoding [10, 13, 14]. The encoding turned out to lead to queries and universal plans of significantly larger size than encountered in any real-life relational scenarios (as a typical data point, universal plans of 300 atoms were obtained by chasing queries of 20 atoms). Both the chase and the backchase implementation were engineered to scale, and the feasibility of the method was proven in a battery of experiments [10, 12].

Relationship to data integration and non-equivalent rewritings. The C&B algorithm looks for rewritings that are equivalent (retrieve the same answers as the original query). Under this semantics, it is more general than previously known algorithms for rewriting using views, because it additionally takes into account general constraints (tgds and egds). In many integration scenarios however, there is no equivalent rewriting and one is content to approximate the original query q by finding a *maximally-contained rewriting*. Significant research has been carried out on algorithms which find such rewritings for conjunctive queries. Contained rewritings are unions of conjunctive queries expressed exclusively in terms of the views and contained in q [15, 21]. Maximally-contained rewritings are contained rewritings which contain any other contained rewriting of q, thus being the best “under-approximation” of q using the views. The problem was generalized in [7] to replace views with schema mapping constraints from the source schema to the target schema, also allowing constraints on the target schema. [22] generalizes the setting even further, allowing schema mappings in both directions, and settling the problem by charting its decidability boundaries and providing tight complexity bounds.

Although in its basic form the C&B algorithm returns only *equivalent* rewritings, it turns out that a simplified version acts as a dual to the algorithms for finding maximally-contained rewritings [21], by providing an alternate approximation: the *minimally-containing rewriting*. A containing rewriting of q is a conjunctive query against the views which contains q. A minimally-containing rewriting is a containing rewriting which is contained in any other containing rewriting of q. It is thus the best “over-approximation” of q. The simplified algorithm is the following:

1. Chase q and obtain the universal plan U.
2. Restrict the body of U only to the vocabulary of views, obtaining a query M.
3. If M is safe (i.e., its head variables appear in the body), output M, otherwise output “no containing rewriting of q exists”.

This simplification of C&B skips the backchase minimization stage. The following result states that the algorithm is sound and complete for finding the minimally-containing rewriting, which is unique up to equivalence:

Theorem 4 *Assume that the chase of q terminates. Then q admits a minimally-containing rewriting if and only if the simplified C&B algorithm outputs such a rewriting. Moreover, the minimally-containing rewriting is unique up to equivalence.*

Relationship to data exchange There are several interesting parallels (and differences) between the C&B method and the formalism for data exchange that was developed in [18, 19].

The data exchange problem is the problem of materializing an instance of a target schema based on an instance of a source schema, and based on a set of source-to-target constraints, representing the mapping between the two schemas.

First of all, both methods make use of the chase in a fundamental way. The C&B method applies the chase to construct the universal plan, while in data exchange, the chase is applied on the source instance to construct a *universal solution*. Philosophically, the concepts of universal plan and universal solution are somewhat similar and play equally important roles. The universal plan defines the space of all minimal reformulations while the universal solution is the "best" representative for the space of all possible target instances (or, solutions).

Second, both methods use minimization: in C&B, to generate all the minimal reformulations, in data exchange, to compute the smallest universal solution (the *core* of the universal solutions [19]). In C&B, minimization is performed under constraints and we look for multiple and non-isomorphic reformulations that are minimal under constraints. In contrast, in data exchange there is only one core of the universal solutions (up to isomorphism). This core is defined independently of the constraints and represents the minimal form of a universal solution, under homomorphisms which preserve the values that appear in the source instance. Finally, another (important) difference is the complexity of the minimization process in the two cases. In data exchange, computing the core of the universal solutions has polynomial-time algorithms in several cases of practical relevance [19, 20]. In the more general setting of C&B, minimization is exponential (NP-hard even without constraints, when it becomes tableau minimization [9]).

7 Conclusion

Many classical database problems such as semantic optimization (i.e. rewriting using semantic constraints), minimization, rewriting using views, equivalent query reformulation in data publishing and integration, are particular instances of query reformulation under constraints. While the general reformulation problem is undecidable, the least restrictive known conditions which are sufficient to guarantee decidability (namely weak acyclicity of the constraint set) hold in numerous practical scenarios. Under these conditions, C&B is a sound and complete algorithm, thus providing a uniform solution to the above problems (with applicability to object-oriented and XML settings). Our experiments show that, with careful engineering of the chase and backchase phases, the C&B method is viable in practice. An online demo of the C&B method can be found at `http://cb.ucsd.edu`.

References

[1] S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *PODS*, pages 254–263, 1998.

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

[3] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. Subrahmanian. Query Caching and Optimization in Distributed Mediator Systems. In *SIGMOD*, pages 137–148, 1996.

[4] C. Beeri and Y. Kornatzky. Algebraic Optimisation of Object Oriented Query Languages. *TCS*, 116(1):59–94, 1993.

[5] C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *JACM*, 31(4):718–741, 1984.

[6] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of Programming with Collection Types. *TCS*, 149(1):3–48, 1995.

[7] A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Data Integration under Integrity Constraints. In *CAiSE*, pages 262–279, 2002.

[8] R. G. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez, editors. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

[9] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*, pages 77–90, 1977.

[10] A. Deutsch. *XML Query Reformulation Over Mixed and Redundant Storage*. PhD thesis, Dept. of Computer and Information Sciences, University of Pennsylvania, 2002.

[11] A. Deutsch, L. Popa, and V. Tannen. Physical Data Independence, Constraints and Optimization with Universal Plans. In *VLDB*, pages 459–470, 1999.

[12] A. Deutsch and V. Tannen. MARS: A System for Publishing XML from Mixed and Redundant Storage. In *VLDB*, pages 201–212, 2003.

[13] A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints. In *ICDT*, pages 225–241, 2003.

[14] A. Deutsch and V. Tannen. XML Queries and Constraints, Containment and Reformulation. *TCS*, 336(1):57–87, 2005.

[15] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.

[16] R. Fagin. Horn Clauses and Database Dependencies. *JACM*, 29(4):952–985, Oct. 1982.

[17] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *ICDT*, pages 207–224, 2003.

[18] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.

[19] R. Fagin, P. G. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. *ACM TODS*, 30(1):174–210, 2005.

[20] G. Gottlob. Computing Cores for Data Exchange: New Algorithms and Practical Solutions. In *PODS*, 2005.

[21] A. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, pages 270–294, 2001.

[22] C. Koch. Query Rewriting with Symmetric Constraints. In *Proceedings of FoIKS (LNCS 2284)*, pages 130–147, 2002.

[23] K. Lellahi and V. Tannen. A calculus for collections and aggregates. In E. Moggi and G. Rosolini, editors, *LNCS 1290: Category Theory and Computer Science (Proceedings of CTCS'97)*, pages 261–280, 1997.

[24] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[25] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *PODS*, pages 95–104, 1995.

[26] A. Y. Levy, A. Rajamaran, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, pages 251–262, 1996.

[27] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing Implications of Data Dependencies. *ACM TODS*, 4(4):455–469, 1979.

[28] L. Popa. *Object/Relational Query Optimization with Chase and Backchase*. PhD thesis, Dept. of Computer and Information Sciences, University of Pennsylvania, 2000.

[29] L. Popa, A. Deutsch, A. Sahuguet, and V. Tannen. A Chase Too Far? In *SIGMOD*, pages 273–284, 2000.

[30] L. Popa and V. Tannen. An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In *ICDT*, pages 39–57, 1999.

[31] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.

[32] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. Querying XML Views of Relational Data. In *VLDB*, pages 261–270, 2001.

[33] O. Tsatalos, M. Solomon, and Y. Ioannidis. The GMAP: A Versatile Tool for Physical Data Independence. *VLDB Journal*, 5(2):101–118, 1996.

A Some Definitions

We review the standard definitions of conjunctive queries, homomorphisms, containment mappings and chase.

A *conjunctive query* q over a schema Σ is an expression of the form $q(\mathbf{x}) :- \phi(\mathbf{x}, \mathbf{y})$ where $\phi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas (i.e., relational atoms, also called *subgoals*) over Σ. We follow the usual notation and separate the atoms in a query by commas. We call $q(\mathbf{x})$ the *head* and $\phi(\mathbf{x}, \mathbf{y})$ the *body*. We use a notation such as $\mathbf{x}$ for a vector of variables $x_1, \ldots, x_k$ (not necessarily distinct). Every variable in the head must appear in the body(i.e., the query must be *safe*). The set of variables in $\mathbf{y}$ is assumed to be existentially quantified.

Given two conjunctions $\phi(\mathbf{u})$ and $\psi(\mathbf{v})$ of atomic formulas, a *homomorphism* from $\phi(\mathbf{u})$ to $\psi(\mathbf{v})$ is a mapping h from the set of variables in $\mathbf{u}$ to the set of variables in $\mathbf{v}$ such that for every atom $R(u_1, \ldots, u_n)$ of ϕ, the atom $R(h(u_1), \ldots, h(u_n))$ is in ψ. Given two conjunctive queries $q_1(\mathbf{x}) :- \phi(\mathbf{x}, \mathbf{y})$ and $q_2(\mathbf{x}') :- \psi(\mathbf{x}', \mathbf{y}')$, a *containment mapping* from q_1 to q_2 is a homomorphism h from $\phi(\mathbf{x}, \mathbf{y})$ to $\psi(\mathbf{x}', \mathbf{y}')$ such that $h(\mathbf{x}) = \mathbf{x}'$. A classical result [9] states that a necessary and sufficient condition for the containment (under all instances) of a conjunctive query q_1 into a conjunctive query q_2 is the existence of a containment mapping from q_2 to q_1.

Assume a conjunctive query $q(\mathbf{x}) :- \phi(\mathbf{x}, \mathbf{y})$ and a tgd t of the form $\forall \mathbf{u}(\alpha(\mathbf{u}) \rightarrow \exists \mathbf{v} \beta(\mathbf{u}, \mathbf{v}))$. Assume without loss of generality that $\mathbf{v}$ and the query have no variables in common. The chase of q with t is applicable if there is a homomorphism h from $\alpha(\mathbf{u})$ to the body of q, and moreover, if h cannot be extended to a homomorphism h' from $\alpha(\mathbf{u}) \wedge \beta(\mathbf{u}, \mathbf{v})$ to the body of q. In that case, a *chase step* of q with t and h is a rewrite of q into $q'(\mathbf{x}) :- \phi(\mathbf{x}, \mathbf{y}) \wedge \beta(h(\mathbf{u}), \mathbf{v})$.

Similarly, we can define a chase step with an egd. Assume a conjunctive query q as before and an egd e of the form $\forall \mathbf{u}(\alpha(\mathbf{u}) \rightarrow (u_1 = u_2))$. The chase of q with e is applicable if there is a homomorphism h from $\alpha(\mathbf{u})$ to $\phi(\mathbf{x}, \mathbf{y})$ so that $h(u_1)$ and $h(u_2)$ are not the same variable. In that case, a *chase step* of q with e and h is a rewrite of q into a query q' which is the same as q except that all occurrences of the variable $h(u_1)$ (in the head and in the body) are replaced by the variable $h(u_2)$.